Representation of Industrial Knowledge - as a Basis for Developing and Maintaning Product Configurators

Haug, Anders

Publication date:
2008

Document Version
Publisher's PDF, also known as Version of record

Representation of Industrial Knowledge

- as a Basis for Developing and Maintaining Product Configurators

A PhD thesis by

Anders Haug

Representation of Industrial Knowledge

- as a Basis for Developing and Maintaining Product Configurators

A PhD thesis by

Anders Haug

2007
Department of Manufacturing Engineering & Management
Technical University of Denmark

Representation of Industrial Knowledge
- as a Basis for Developing and Maintaining Product Configurators

Supervisor:
Associate Professor, Lars Hvam
Department of Manufacturing Engineering & Management
Technical University of Denmark

Published by:
Department of Manufacturing Engineering & Management
Technical University of Denmark
Produktionstorvet, bygning 424
DK-2800 Kgs. Lyngby

ISBN: 978-87-91035-67-8

Preface

The present Ph.D. thesis is a result of a three-year work process carried out at the Department of Manufacturing Engineering and Management, Technical University of Denmark. The project was initiated in January 2005 and completed in December 2007.

In contrast to what I was often told at the beginning of my PhD project, I actually found the process of being a PhD student to be a very pleasant experience, where I thoroughly enjoyed the inspirational and stimulating challenges associated with the process. However, to a considerable degree, this is a result of the support by a long list of people who have been helpful during the process. I therefore offer my thanks to the following persons:

- Former PhD students of Lars Hvam, Benjamin Hansen and Jesper Riis for feedback and discussions on many of my early papers.
- My PhD colleagues, Klaes Ladeby, Gudmundur Oddsson, Tim Christensen and Thomas Petersen for their pleasant company on journeys to conferences/PhD-courses and for interesting discussions.
- Assistant Professor Kasper Edwards, for always being willing to give his opinion regarding various research problems.
- Professor Mogens Myrup Andresen, for taking time to discuss the field of engineering design.
- Associate Professor Kai Jørgensen, for inspiring discussions during the course of my PhD project.
- Associate Professor Bjarne Poulsen, for his collaboration in my research on configuration documentation systems.
- My father, Karsten Haug, for proof-reading paper abstracts and for his efforts to improve my English.
- Correspondent Christina Scheel Christiansen, for her enormous work of proof-reading my papers and for patiently helping me improve my English.
- English language editor, Mary Bille, for taking on the proofreading of the thesis on short notice.
- Cand.scient.pol. Søren Bisgaard for discussions and feedback regarding the scientific part of the thesis.
- The many persons from the studied companies who have without hesitation participated in interviews and given me access to information about their configuration projects, in particular F.L. Smidth, GEA Niro, American Power Conversion and Altan.dk.
- My supervisor, Associate Professor Lars Hvam, for ensuring me excellent conditions while I elaborated my PhD, such as setting up contacts with relevant companies for case studies, providing possibilities to test my propositions, supporting my choice of the directions of my research and providing inspiration to define research problems.

Also, I would like to thank my family and friends for making sure I did not disappear into my theoretical bubble during the course of the project. Finally, I also offer my thanks to all those I may have forgotten, who also have played an important role in the process.

Lyngby, December 2007,
Anders Haug

Abstract

A product configurator is a software-based expert system that supports the user in the creation of product specifications by restricting how different components and properties may be combined. The use of product configurators has for several years provided many engineering-oriented companies such benefits as: shorter lead times, improved quality of product specifications, preservation of knowledge, use of fewer resources for specifying products, optimized products, less routine work, improved certainty of delivery, and less time needed for training new employees. Unfortunately, not all configuration projects are successful, but in fact many fail or experience great problems during the course of the project. An important factor for the success of a configuration project is the quality of the methods, techniques and tools applied when extracting, representing and documenting relevant domain knowledge. Despite this fact, research in the knowledge acquisition process of configuration projects is an area that has been much neglected till now. Therefore, this thesis deals with some of the most important aspects of the knowledge acquisition process in configuration projects by answering seven research questions in nine papers, produced during the course of the PhD project. The questions are grouped under three topics: domain expert knowledge; knowledge representation techniques; and documentation of configuration knowledge.

The thesis takes its point of departure in analysing existing literature, after which research questions are defined, a frame of reference established and the scientific approach outlined. Next, the main contribution of the PhD project is described, namely the papers that are part of the thesis, starting with analysis of the process in which domain experts provide relevant information to knowledge engineers. The process is investigated by analysing the role of tacit knowledge in configuration projects and by proposing a classification of the kinds of information involved in this process. The thesis then investigates how the information retrieved from domain experts can be represented in analysis and design models. To solve inadequacies of an existing graphic knowledge representation technique, the thesis proposes a representation technique that combines the existing technique with tables and other modelling constructs. Next, the two most commonly applied graphic knowledge representation techniques in configuration projects are investigated by analysing their mutual strengths and weaknesses. Having clarified the nature of these strengths and weaknesses, a new layout principle is proposed that combines the advantages of both notation techniques. To deal with cases where graphic models with overlapping content are to be maintained, the thesis proposes and tests a modelling principle that allows maintenance of models with overlapping content in a common model. Finally, the thesis investigates how knowledge in configuration projects can be documented, from a software perspective. This is done by proposing definitions of the modelling techniques that a software-based documentation system should support. To test the definitions, a software prototype is developed.

In conclusion, this thesis provides new insights into the knowledge acquisition process of configuration projects and several new modelling techniques and principles. The contributions provide an improved basis for future research in product configuration and for the companies that engage in configuration projects.

Resume

En produktkonfigurator er et softwarebaseret ekspertsystem som understøtter brugerne i skabelsen af produktspecifikationer, ved at begrænse hvordan forskellige komponenter og egenskaber må kombineres. Brugen af produktkonfiguratorer har i flere år bidraget til mange gevinster for ingeniørorienterede virksomheder, så som kortere gennemløbstider, forbedret kvalitet af produktspecifikationer, fastholdelse af viden, mindre ressourceforbrug, bedre leveringssikkerhed, og mindre tidsforbrug på uddannelse af nye medarbejdere. Desværre er det ikke alle konfigureringsprojekter som er succesfulde, men mange projekter fejler eller oplever store problemer. En afgørende faktor i forhold til et konfigureringsprojekts succes er kvaliteten af de metoder, teknikker og værktøjer der anvendes, når ekspertviden udtrækkes, repræsenteres og dokumenteres. På trods af dette forhold, er forskning i vidensakkvisition i konfigureringsprojekter et område som har været relativt forsømt til nu. Denne afhandling tager derfor fat i nogle af de vigtigste aspekter af dette område ved at besvare syv forskningsspørgsmål i ni artikler, produceret under ph.d.-projektet. De syv spørgsmål er grupperet på emnerne domæneekspertviden, videnrepræsentation, og dokumentation af konfigureringsviden.

Afhandlingen tager sit afsæt i en analyse af eksisterende litteratur, på hvilken baggrund forskningsspørgsmål formuleres, en referenceramme opsættes og en videnskabelig tilgang defineres. Efterfølgende er afhandlingens væsentligste bidrag, de inkluderede artikler i afhandlingen, beskrevet. Disse starter med en analyse af processen, hvor domæneeksperter leverer relevante informationer til en vidensingeniør. Processen undersøges ved at beskrive tavs videns rolle i konfigureringsprojekter og ved at foreslå en klassifikation af den information der er involveret i denne proces. Afhandlingen undersøger derefter, hvordan den leverede information kan repræsenteres i analyse- og designmodeller. For at løse uhensigtsmæssigheder ved en eksisterende grafisk videnrepræsentationsteknik foreslår afhandlingen en ny grafisk notationsteknik som kombinerer den eksisterende teknik med tabeller og andre modelleringskoncepter. Dernæst er to af de mest applicerede grafiske videnrepræsentationsteknikker undersøgt ved at sammenligne deres indbyrdes styrker og svagheder. På baggrund af at have klarlagt dette forhold foreslås et nyt layoutprincip, som kombinerer fordelene fra begge disse notationsteknikker. For at forbedre håndteringen af grafiske modeller med overlappende indhold i cases, hvor disse skal vedligeholdes, foreslår og tester afhandlingen et nyt modelleringsprincip, som tillader at separate modeller med overlappende indhold kan vedligeholdes i en fælles model. Endelig undersøger afhandlingen, hvordan viden i konfigureringsprojekter kan dokumenteres fra et softwareperspektiv. Dette gøres ved at foreslå definitioner af de modelleringsteknikker, som et softwarebaseret dokumentationssystem skal understøtte. Definitionerne testes ved at udvikle en softwareprototype.

Som opsummering producerer denne afhandling nye indsigter i vidensakkvisitionsprocessen i konfigureringsprojekter samt adskillige nye modelleringsteknikker og principper. Afhandlingen bidrager hermed til et forbedret grundlag for fremtidig forskning i produktkonfigurering og for virksomheder som påbegynder konfigureringsprojekter.

Table of contents

Chapter 1: Introduction **1**
1.1 Background 1
1.2 Purpose 1
1.3 Focus 2
1.4 Structure of the thesis 3
Chapter 2: Research context **5**
2.1 A brief resume of the history of configurators 5
2.2 Configuration research 5
2.3 Discussion of existing research 11
Chapter 3: Research questions **15**
Chapter 4: Frame of reference **17**
4.1 Configurators 17
4.2 Models 21
4.3 Knowledge representation 24
4.4 The CPM-procedure 32
4.5 Mass customization at ETO companies 35
Chapter 5: Scientific approach **37**
5.1 Basic concepts 37
5.2 Scientific approach of existing research 38
5.3 Alternative scientific approaches 42
5.4 Chosen scientific approach 47
Chapter 6: Presentation of papers **53**
6.1 Overview of the papers 53
6.2 A) Domain expert knowledge 54
6.3 B) Knowledge representation techniques 57
6.4 C) Documentation of configuration knowledge 65
Chapter 7: Conclusion **71**
7.1 Research questions 71
7.2 Methodological reflections 76
7.3 Summary of contributions 76
7.4 Future research 77
References **79**
Appendix 1: Reflections on the transition from ETO to MC **87**
Appendix 2: List of publications **95**
Appendix 3: Contribution to appended papers **97**
Appendix 4: Usability experiment **99**
Appended papers **101**

Chapter 1: Introduction

1.1 Background

A product configurator is a software-based expert system that supports the user in the creation of product specifications by restricting how different components and properties may be combined. The use of product configurators has for several years provided such benefits to many engineering-oriented companies as: shorter lead times, improved quality of product specifications, preservation of knowledge, use of fewer resources for specifying products, optimized products, less routine work, improved certainty of delivery, and less time needed for training new employees (e.g. Ardissono et al., 2003; Hvam, 2004; Hvam et al., 2007a; Forza and Salvador, 2002b; Forza and Salvador, 2007). A good example of how significant the effects from a configuration project can be is the case of F.L. Smidth, manufacturer of large processing plants for cement production. F.L. Smidth started operation of their configurator for budget quotes in 1999, and the company has since experienced significant reductions of parameters related to the creation of budget quotations. For instance, the average engineer resources used on a quote was reduced from 5 man-weeks to 0,2 man-weeks, and the time needed for the creation of a quote was reduced from 2-5 weeks to 1-3 days (Hvam, et al, 2006).

1.2 Purpose

Unfortunately, not all configuration projects are successful; in fact, many fail or experience great problems during the course of the project. A major cause for these problems in the many cases studied at Department of Manufacturing Engineering and Management at the Technical University of Denmark is underestimation of the work required to carry the project through. In this context, one of the most challenging and time-consuming assignments for a configuration project is to represent the relevant domain knowledge to be implemented in the configurator (Edwards et al., 2005; Sabin and Weigel, 1998; Hvam et al., 2007a; Hansen et al, 2003; Forza and Salvador, 2007). It can be challenging in itself just to convert information from domain experts into models, but in addition, such models have to be of a high level of clarity, correctness and extensiveness in order to avoid problems when implementing them into the knowledge base of the configurator. For this reason, the quality of the methods, techniques and tools applied when extracting, representing and documenting relevant domain knowledge is of the greatest importance. Despite this fact, research in the knowledge acquisition process in configuration projects has been much neglected so far. As pointed out by Hvam et al.:

> "...the area of knowledge acquisition in the context of product modelling and product configuration is quite untouched, although generic methods exist for knowledge acquisition... Hence, there is a need for new and more specific methods for how the knowledge of complex products is formalized and modelled into a product model that can serve as a basis for a product configuration system." (Hvam et al., 2006)

This lack of knowledge regarding knowledge acquisition in configuration literature is attacked by this PhD thesis. In brief, the purpose of the thesis can be formulated as follows:

> "To improve the methods, techniques, tools and understandings that form the basis for representation of configuration knowledge"

With this defined purpose, the thesis targets two audiences, namely industry and academia. The contribution to industry is mainly the production of better methods, techniques and tools for the development and maintenance of product configurators. The contribution to academia is associated with such contributions as providing explanations of phenomena in configuration projects, new models for understanding reality, and evidence for various claims.

1.3 Focus

The focus of this thesis is representation of configuration knowledge as a basis for building and maintaining product configurators. This includes more than just knowledge representation techniques; it also includes understanding the information that forms the basis for representing domain knowledge, and definitions of the software that can support the elaboration and maintenance of knowledge representations. This is illustrated in figure 1, which shows how a configuration project is often carried out in principle. The first four processes within the box are the main focus of this thesis. The dotted line from analysis to design model illustrates that the distinction between analysis and design model is not necessarily explicitly defined. This can in practice merely be a principle distinction, since often the design models are just more formalized and detailed versions of analysis models. The dotted lines to the documentation system symbolize that it differs what kind of models are placed in a documentation system, if any. Also, the figure is obviously just an idealized description, which, however, in most cases with a structured approach towards the development of a configurator, reflects the process. The difference in practice can be that the domain experts themselves produce the needed conceptual models, or that domain knowledge is implemented directly into the configurator. The latter, however, would be extremely difficult to carry out in projects of some complexity.

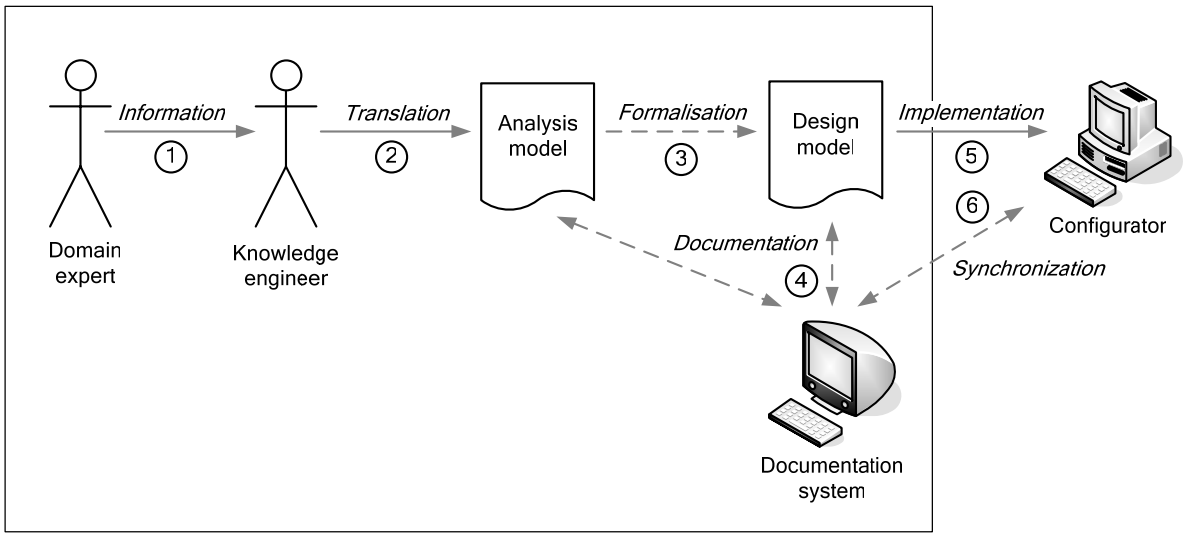


Figure 1: Focus of thesis

The purpose of improving the basis for representation of configuration knowledge implies the need for answers to at least three overall questions:

- What is it for information that the knowledge engineer retrieves? (Process 1)
- How can the retrieved information be represented efficiently? (Process 2 and 3)
- How can the elaboration and maintenance of knowledge representations efficiently be supported by software? (Process 4)

These three preliminary questions are to be further detailed after having investigated what configuration literature has to offer on these points. It should be noted that these three questions should not be seen as the only relevant questions in the given context. For instance, the methods used for eliciting domain expert knowledge and the sub-process during which a knowledge engineer interprets the information retrieved from domain experts are topics that later could be investigated in more detail.

It should also be emphasized that the focus of the thesis is on products of some complexity, i.e. most often involving industrial engineering companies engaged in configuration projects in order to support some processes with a configurator. Such companies are primarily labelled ETO

(Engineering-To-Order) companies and to some extent also ATO (Assembly-To-Order) and MTO (Make-To-Order)[1] companies.

Finally, the thesis builds on the assumption that has formed the basis for more than ten years of configuration research at my particular department, namely that although general software development procedures, methods and techniques can be useful in a configuration context, product configuration is a special case of software development projects that is best carried out by the use of procedures, methods and techniques with this particular focus.

1.4 Structure of the thesis

The thesis begins with a résumé of the research context of which the contributions of this PhD thesis should be seen as an extension, i.e. the relevant configuration literature. This overview provides the basis for the formulation of the research questions and a frame of reference. In the following chapter, the scientific approach of the thesis is defined. The next chapter summarizes the main part of the PhD thesis, namely the nine appended papers. Finally, the produced results of this thesis are summarized in the concluding chapter. The overall structure and main tasks of the thesis are shown in figure 2.

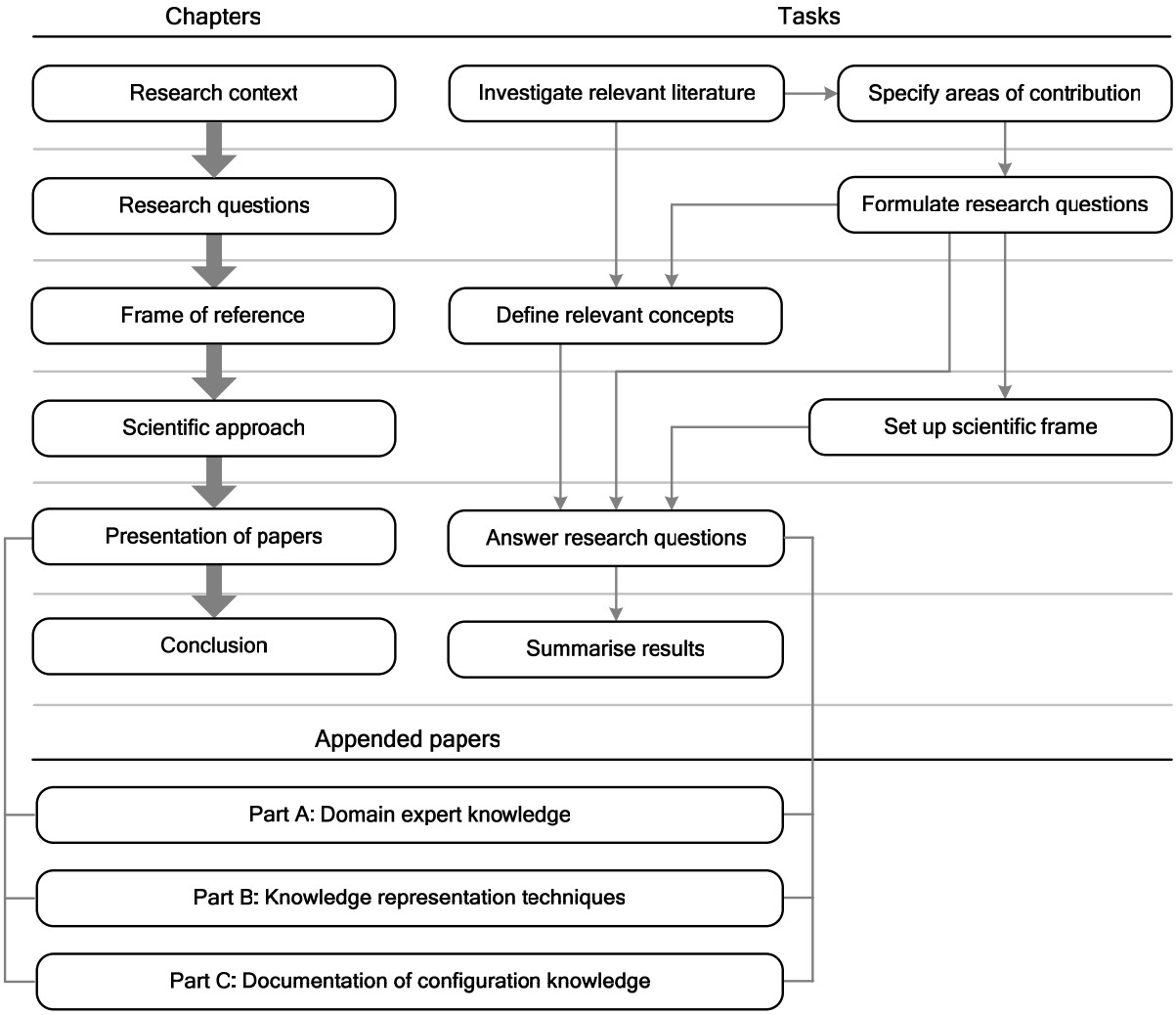


Figure 2: Structure of the thesis

[1] 'MTO' carries a different meaning in the literature, used in this context as in Forza et al., 2006.

The development in the appended papers and the relationships between the three topics that they deal with are illustrated in figure 3.

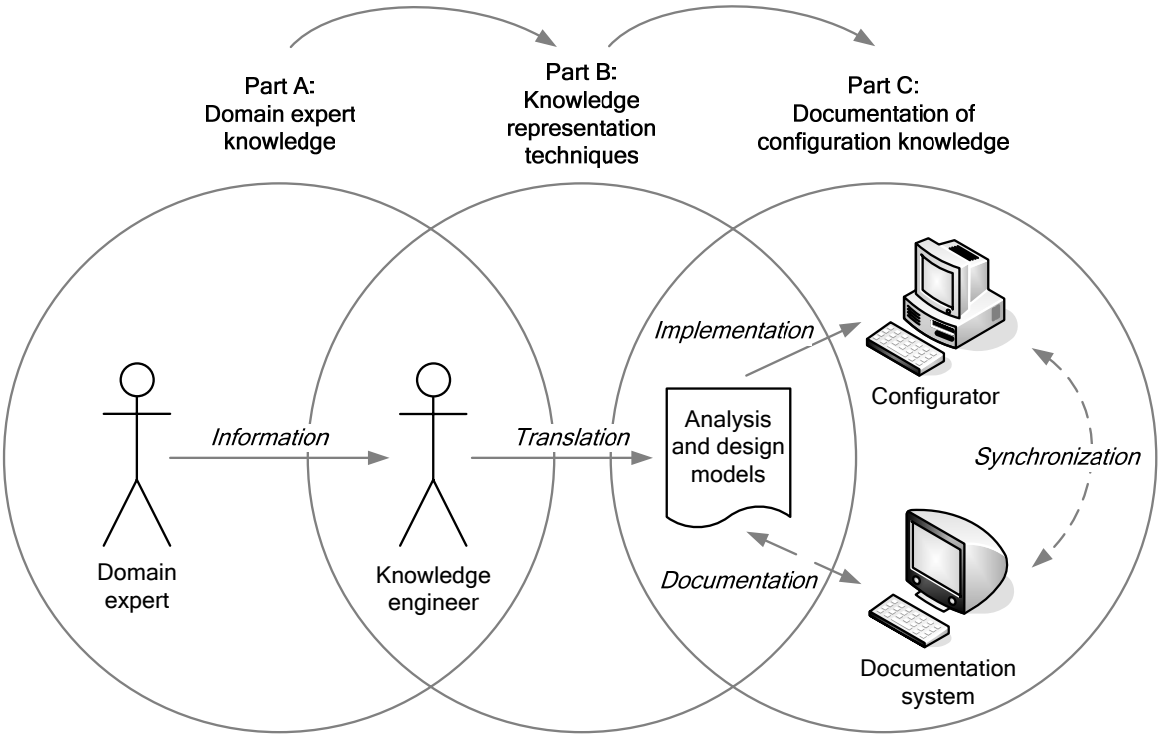


Figure 3: Relationships between the topics of the appended papers

Chapter 2: Research context

2.1 A brief resume of the history of configurators

The first product configurators that emerged were rule-based. One of these was 'R1', which was developed in the late 1970s by Digital Equipment Corporation for configuring computer systems (McDermott, 1982). Later a derivative of this system, named XCON, was developed. XCON served as the ancestor for a family of expert systems used for configuration of hardware-related products at Digital (Stumptner, 1997). Digital's R1/XCON is often described as the first really successful configurator (e.g. Barker and O'Connor, 1989).

In the 1980s, more companies began to apply configurator technology. In the beginning, these configurators were rule-based systems without a division between knowledge base and control strategy. However, the maintenance of the rules-based configurators was difficult and time consuming. Responding to these problems, Mittal and Frayman (1989) proposed a generic and domain-independent model for configuration tasks in the form of a constraint satisfaction problem (CSP). In contrast with the procedural approach, this perspective implied that configuration problems could be modelled as problems consisting of a finite set of variables with a finite set of possible values, where constraints restrict variable combinations and allowed values. Later, further developments of CSPs emerged, such as 'dynamic CSPs' (DCSPs) (Mittal and Falkenhainer, 1990), 'generative CSPs' (GCSPs) (Stumptner and Haselböck, 1993) and 'conditional CSPs' (CCSPs) (Sabin and Freuder, 1998).

Since their emergence, configuration expert systems have been built for configuring a great variety of industrial products such as: operating systems (Haugeneder et al., 1985); elevators (Marcus et al., 1988); circuit boards (Birmingham et al., 1988); aircraft cabins (Kopisch and Günter, 1992); telecommunication systems (Wright et al., 1993; Stumptner at al., 1994; Fleischanderl et al. 1998); computer systems (Sabin and Weigel, 1998); electronic switchboards (Hvam et al., 2002); power transformers (Forza and Salvador, 2002a); mould blocks (Forza and Salvador, 2002b); data centre systems (Hvam, 2006a); cement plants (Hvam, 2006b); and induction motors (Forza et al., 2006).

2.2 Configuration research

Much research related to product configuration has been carried out. In this chapter, a review of relevant configuration literature is presented. The chapter should not be perceived as the theoretical foundation for the PhD thesis. Instead, the purpose this chapter is to provide an overview of configuration literature in order to define the research questions and relevant concepts. The review provided in this chapter is by no means complete, but it includes some of most important literature for this PhD project. The review has two major delimitations. First, the chapter is delimited to literature that deals explicitly with product configuration/-configurators. This delimitation has been applied in order to avoid that the scope of the review become unreasonably extensive. On the other hand, this does not mean that literature from other fields is not reviewed in this PhD thesis; such literature is mentioned in the papers when used. This includes among others: mass customization, knowledge engineering/acquisition, engineering design, knowledge management, usability, and object-oriented modelling techniques. Also in chapter 4, 'Frame of reference', some of these areas are touched on. The second delimitation of the review is that it primarily includes ISI-indexed journal papers, based on the assumption that these represent to a great extent the most significant research within this field.

2.2.1 Research groups

According to Soininen at al. (1998), most of the research on configuration has focused on problem-solving methodologies, such as constraint satisfaction, resource-based configuration and propose-and-revise approaches. However, the emphasis of configuration research on other aspects has increased since the statement by Soininen et al., which is shown in the review of configuration research in the

following sections. When focusing on journal publications, most of the relevant research for this PhD (hereby excluding most software technical research) can be organized into four groups according to from where/ whom the research has emerged:

- University of Klagenfurt
- Helsinki University of Technology
- Forza and Salvador
- Technical University of Denmark

Overall, these groups can be perceived as creating the context to which this PhD thesis aims to contribute. In the following, the research conducted by these groups is briefly summarized, followed by a discussion of their relevance for this PhD project.

2.2.2 University of Klagenfurt

At the University of Klagenfurt, Austria, Alexander Felfernig has published a long list of journal papers about configuration, together with other researchers. Being one of the few configuration research groups that deals with the adaptation of graphic modelling techniques for the description of configuration knowledge, this literature is an important basis for this PhD project.

Felfernig et al. (2000a) show how to use UML as notation to simplify the construction of logic-based descriptions of relevant domain knowledge in configuration projects. The key idea of the approach is first to extend the static UML model with commonly used configuration concepts[2], and second, to create a definition of how these concepts are mapped to a configuration language. This means that the UML models can automatically be translated into logical sentences that can be used by a general inference engine for solving configuration problems. The choice of UML in this approach is based on arguments of the wide application in industrial software development and the positive experiences with the use of UML designs for validation by technical domain experts. Felfernig et al. (2000b) deal with distributed configuration problem solving. To facilitate distributed configuration of customizable products, they propose the use of cooperating configuration agents that are capable of managing requests and posting configuration subtasks to remote configuration agents. To integrate different knowledge representation formalisms, they apply commonly used configuration concepts to design shareable ontologies that can be interpreted by other agents. The concepts are defined as UML stereotypes that can be translated automatically into a configuration agent's knowledge representation. Felfernig et al. (2000c) show how to apply UML for designing configuration knowledge bases that includes both component structure and functional structure. Also here, the underlying concept is that the created UML models can be translated automatically into executable logical representations in configurators. Felfernig et al. (2000d) introduce the notion of contextual diagrams to cope with the complexity of configuration knowledge (using UML class diagrams). The idea is that domain experts most often think in terms of contexts, and therefore the approach should provide a more intuitive way of modelling configuration knowledge. The proposed concept is that from a root context diagram and the preconditions of a context (formulated by a designer) new contextual diagrams can be derived. These new contextual diagrams include only the relevant elements in a given context. The point of departure of Felfernig et al. (2001) is in the earlier described idea of using UML for the creation of models that can be transferred automatically into a configurator knowledge base. Based on this, they deal with the employment of model-based diagnosis techniques to debug faulty configuration knowledge bases, more specifically to detect unfeasible requirements and to reconfigure knowledge bases. Felfernig et al. (2002) describe an approach for integrating web-based sales systems with complex customizable products/services by making use of descriptive representation formalisms of the Semantic Web.[3] They show that when using UML for knowledge acquisition, it is possible to provide a set of rules that transforms UML models into configuration knowledge bases specified by languages

[2] Classes: component types, resources, and ports. Relationships: connections, compatibility, requires, produces, and consumes. Configuration ontologies are discussed in more detail in chapter 4 of this thesis.

[3] An extension of the World Wide Web, where web content can be expressed in a formal language that can be read by software agents, which permits these to use information more easily.

that represent a foundation for potential future description standards for web services. Based on their earlier work, Felfernig et al. (2003) show how a description logic-based definition of a configuration problem would be represented with consistency based definitions. Thereby, two major streams within the field of knowledge based configuration are joined, and problems can be transferred between different representation principles. Felfernig et al. (2004) show that automated support of debugging processes of knowledge bases can be achieved through the use of consistency-based diagnosis techniques. Based on a formal definition of consistency-based configuration, they present a framework that allows diagnosis of configuration knowledge bases.

2.2.3 Helsinki University of Technology

From the Helsinki University of Technology in Finland a significant amount of configuration research has emerged, much of which, however, with a software technical focus.

Soininen at al. (1998) present a generalized ontology of product configuration, which should be seen as a step towards an ontology of configuration in general. The ontology includes a set of concepts for representation of configuration knowledge, i.e.: components, attributes, resources, ports, contexts, functions, constraints, and relationships. Soininen and Niemela (1999) propose a rule-based language for product configurators named CRL (Configuration Rule Language). The defined language includes formal definitions for main product configuration concepts, including: configuration models, requirements, configurations, valid configurations, and configurations that satisfy requirements. The rule language has been tested on a small configuration problem. Männistö et al. (2001) have the basic assumption that the after-sales support of products with a large number of variants implies that it is difficult to extract meaningful data from the product instances with traditional BOM information. Based on the assumption that data structures suitable for manufacturing are not the most suitable for after-sales, they propose a modelling methodology that allows modelling of product instances on multiple abstraction levels. Kojo et al. (2003) take a point of departure in that the use of software product families (also known as software lines) as a means for increasing efficiency of software development. Therefore, they propose an approach for modelling that evolves software product families based on a subset of the ontology of product configuration knowledge from Soininen at al. (1998). Raatikainen et al. (2004) present results of a case study undertaken in two companies that develop and deploy configurable software product families. Their study shows that the characteristics of the configurable software product families were remarkably similar in spite of very different companies, products, and application domains. They argue that their study shows that it is feasible to systematically develop families of software and manage the variability within the software family. Asikainen et al. (2004) present an approach for the modelling of configurable software product families and for automated configuring of product instances. They introduce a (programming) language for the modelling of configurable product software families named 'Koalish'. This language is an extension of the language 'Koala', a "component model and architecture description language, with explicit variation modelling mechanisms". Asikainen et al. (2007) propose a 'domain ontology' for representing the variability in software product families, which they name 'Kumbang'. The semantics of Kumbang are described using natural language and a UML profile.[4] Kumbang supports the modelling from both feature and architecture based approaches, and includes concepts for modelling the interrelations between these two views.

2.2.4 Forza and Salvador

Forza and Salvador have produced a long list of journal papers on the topic of product configuration, and recently also a book. It should be noted that all their research does not necessarily include the use of product configurators, but just the ability to configure products.

Forza and Salvador (2002a) present a case study concerning the implementation of product configuration software in a small manufacturing company that produces mould-bases for plastics moulding and punching-bases for metal sheet punching. The case study shows that the company

[4] UML profiles are collections of stereotypes and tagged values that tailor UML to specific applications, e.g. product configuration.

obtained both a rapid payback of the investment in configuration technology as well as a competitive advantage. It also shows that the configurator can be propagated to departments not directly involved in the implementation, and that the resulting new work flow can also affect the organization of the customers, i.e. inter-firm co-ordination. Forza and Salvador (2002b) present a case study of a small company, which produces voltage transformers. The study indicates that the implementation of a product configuration system has contributed to better product variety management and reduction of the risk of losing strategic competence due to the departure of key employees. However, they emphasize that the introduction of a product configuration system could require significant changes in the organization's order processes and require a high initial investment in terms of man-hours. Salvador and Forza (2004) take their point of departure in the claim that while product configuration has been widely explored from a technical standpoint, there has only been produced anecdotal evidence of the managerial issues that emerge when customization is offered with a need for short delivery times, i.e. a 'customization-responsiveness squeeze'. Their study includes 122 companies from Northern Italy that face customer requirements for customization with short delivery times. Their study indicates that these companies face such difficulties as problems with information flows, repetitive activities, and many errors. They argue that to obtain the advantages of product configuration, changes in the organization and order support systems are needed (such as product configurators). Salvador et al. (2004) investigate how the supply chain of a company should be configured when different degrees of customization are offered. They conduct case analyses of six companies that manufacture mopeds, heavy commercial vehicles, cell phones, multiplexers, microwave ovens, and steam convection ovens, respectively. Their case studies show that the freedom that customers have when specifying a product heavily affects the supply chain configuration. Furthermore, they identify two kinds of supply-chain configurations: one suggests an isomorphism between market characteristics, product structure and supply-chain configuration, while the other is without such strong relationships. Forza et al. (2006) present a case study of a company that produces electric motors. The case shows how the right grouping of components (into kits) has enabled the company to implement a product configurator and to postpone product differentiation along the material flow. Salvador and Forza (2007) take a point of departure in the fact that the process of specifying customizable products can be a burden for customers, which in a worst case can make customers choose more standardized products instead. Therefore, they present a formalization of the underlying principles that allow a company to present their product assortment to a customer. They describe some main strategies for reducing the efforts potential customers have to make when attempting to understand what a company offers and how the products offered can satisfy their requirements.

An important contribution to product configuration literature is the book by Forza and Salvador, 'Product Information Management for Mass Customization' (Forza and Salvador, 2007). They describe the book as the first that provides a holistic recognition of the essential aspects of an IT-supported configuration system (in this context referring to both the software and humans involved in configuration activities). This claim of holism is supported by the fact that the book covers many aspects related to product configuration by dealing with such topics as: configuration processes, configurable products, different approaches to definition of product configurators, classification of configurators, commercial product modelling, technical product modelling, relations between configuration systems and management information systems, selecting a configuration system, operational implications of a configuration system, and organizational implications of a configuration system. However, in spite of their holistic perspective on configuration, Forza and Salvador do not deal much with the knowledge acquisition process or modelling techniques for expressing product knowledge in a configuration project. In their examples of conceptual models, they use rather simple and informally defined notation techniques.

2.2.5 Technical University of Denmark

From the Centre for Product Modelling (CPM) at Department of Manufacturing Engineering and Management at the Technical University of Denmark, much configuration research has emerged in the form of PhD projects, conference papers, journal papers and a recently published book. Since this PhD thesis has been elaborated at CPM, this literature obviously comprises the main basis for the project.

In his PhD thesis from 1994 (Hvam, 1994/1996),[5] Hvam presents a 'procedure for developing systems to support of the specification activities in the company using product models'. The procedure consists of seven phases for the specification of the activities that are to be IT-supported and the development of these systems. These phases are: 1) Analysis of product- and methods-specification tasks; 2) Determination of content and structure for product and product-related models, and purpose, view and context for the next phase; 3) Preparation of OOA model (object-oriented analysis); 4) Preparation of OOD model (object-oriented design); 5) Programming; 6) Implementation and education of users; and 7) Maintenance of the system. This procedure has since been further developed at CPM into a form later described in section 4.4 of this thesis. In the rest of this thesis, this procedure is referred to as the 'CPM-procedure'. The PhD thesis by Jónsdóttir (1998) deals with the use of concurrent engineering and product models in seafood product development. It has as one of its four objectives to analyse which seafood product development tasks can be supported by the CPM-procedure as presented by Hvam (1996). On this point, she concludes that this procedure is useful to some degree, but the benefits of product modelling "are apparently more in terms of support of the product developers in retrieving and reuse of information and knowledge than in automating some of the specification tasks, e.g. configuration of product specifications". The purpose of the PhD thesis by Riis (2003) is to further develop the CPM-procedure. Regarding modifications and contributions in relation to the definitions of Hvam (1994), Riis includes an extended description of the phases of the procedure, particularly the last three. Compared to the CPM-procedure as defined by Hvam (1994), one of the most interesting changes is of phase 2, where the Product Variant Master (PVM) notation (later described in section 4.3) has been included for describing the relevant domain knowledge before moving into the object-oriented phases. Also, this version of the procedure has a more specific focus on the use of standard configurator software. The PhD thesis by Svensson (2003) focuses on customized (build to order) production, and particularly on the management of the specifications related to a specific order, from pre-sales to delivery of a product. Svensson presents seven research questions. One of these especially concerns product configurators, namely how configurators can be used to improve the utilization of data for build-to-order companies. In response to this question it is concluded that a product configurator can be a very attractive investment in that it can produce more uniform and precise specifications; but for the investment to be profitable requires that the company processes are uniform and that production of the products is of a certain volume. The PhD thesis by Hansen (2003) deals with industrial variant specification systems, i.e. a broader focus than merely on configurators. The thesis objectives are: defining the variant specification system; creating a procedure for the development of variant specification system; developing methods for analysis and description of the (variant) specification task; and developing structural descriptive dimensions and examples of structural solution elements, primarily software. The PhD thesis by Malis (2005) deals with the development of product models in company networks. The main contribution of the thesis is a new procedure for development of product models in company networks. This procedure consists of four phases: 1) Network analysis; 2) Preliminary design for the entire network project; 3) Development of parallel product models; and 4) Operation and maintenance. Phase 3 corresponds to the CPM-procedure as defined by Hvam et al. (2004b).

From CPM, several conference and journal papers have emerged since the Hvam PhD project (1994). Most of this research revolves around the CPM-procedure, but with some variance in the focus. From this work, the following papers can be mentioned. Hvam (1998) presents a course outline for learning product modelling techniques. Hvam and Have (1998) deal with the specification system in engineering companies, and present a proposal for a method for restructuring the processes concerning specification of products in different stages of their life cycle. Hvam and Hansen (1999) discuss product modelling as a tool for preparing the engineering activities that deal with the specification of products in different life-cycle phases, illustrated by examples from the companies Alfa Laval Separation and FL. Smidth. Hansen et al. (2003) use experiences from 11 Danish companies to discuss terminologies, concepts and development trends related to IT-automation of specification activities in engineering companies. Hvam (1999), Hvam (2001) and Hvam and Riis (2003) present revised versions of the CPM-procedure. Hvam and Malis (2001) and Hvam et al. (2005) deal with the topic of creating a documentation system to support the CPM-procedure. Finally,

[5] Hvam, 1996 is the English translation of Hvam, 1994

several case studies on the application of the CPM-procedure have been carried out, e.g. American Power Conversion (Hansen and Hvam, 2002; Hvam, 2006b); Demex Electric (Hvam et al., 2002); F.L. Smidth (Hvam et al., 2004a; Hvam, 2006a; Hvam et al., 2006); and multiple companies (F.L. Smidth, GEA Niro, American Power Conversion and Demex Electric) (Hvam, 2004).

Recently much of the experience of CPM has been collected in the book 'Produktkonfigurering' (Hvam et al., 2007a) and the English version 'Product Customization' (Hvam et al., 2007b). The book is built around the latest version of the CPM-procedure (described in section 4.4).

Another important basis for this PhD project is the PETO[6] research project (Edwards et al., 2005), which this PhD thesis utilises by building on the results from the study and by applying the collected data, i.e. transcripts and sound files from the conducted interviews. The purpose of the PETO research project was to analyse and describe development, implementation and maintenance of configurators from economic, technical and organizational perspectives. The project was carried out during the period 2003 to 2004 and includes studies of twelve Danish firms that were using product configurators at the time of the investigation. The PETO report includes several interesting findings, such as:

- The development and implementation of a configurator often has radical consequences for the way resources are used and managed, for which reason expenses and gains can appear other places than where expected at first.
- While configuration projects have a positive effect on the overview of the product assortment and modularity of products, there may be a negative effect on the degree of innovation due to less customer involvement in the development of new products. This can, however, be avoided through different initiatives.
- In some of the investigated companies, the lack of use of structured methods and techniques seems to have resulted in unstructured models, inadequate documentation or poor choice of software.
- Most of the investigated companies did not document the models in their configurator externally, which meant that several of them experienced great problems with maintaining/further developing their configurators, and problems in the daily communication between product developers and configurator developers.
- Most of the investigated companies perceived configuration projects as technical projects and did not recognize that the implementation of a configurator in the organization affects the daily work processes to the same degree as e.g. ERP systems. By perceiving configuration projects as merely technical projects, organizational changes and learning are neglected, which can mean that the organization may refuse to deliver the required information for building the configurator and later refuse to use the system.
- The majority of the investigated configuration projects were delayed, among other reasons because of the interdisciplinary nature of the projects. The creation of a configurator depends on information that is not unambiguous or fully accessible when the project starts; therefore, project plans are continuously being revised as project groups become familiar with the technology.
- Several of the companies investigated had experienced dramatic reductions in lead times, better quality of specifications, less use of resources etc.
- Areas such as better quality, shorter lead times and less use of resources are connected in the sense that improvement in one of these areas results in improvements in all three areas.

In addition to the report, the results of the PETO project have also been communicated in various conference papers (e.g. Møldrup and Møller, 2004; Pedersen and Edwards, 2004; Edwards and Pedersen, 2004; Edwards and Riis, 2004). The PETO project also inspired the creation of the PETO conference; the first was held in 2004, and was merged in 2006 with the IMCM[7] conference, which was held in merged format in 2006 and again in 2007.

[6] PETO: Economic, Technical and Organizational aspects of Product configuration systems
[7] IMCM: International Mass Customization Meeting

2.2.6 Others

Besides the four above-mentioned groups, other journal papers that deal with the development of graphical notations for the modelling of configuration knowledge have been identified. In this literature, authors invent new graphical notations (or fail to describe where their notation originates from), not as the main focus of the paper but often to illustrate a point (e.g. Aldanondo et al, 2000; Chao and Chen, 2001; Magro and Torasso, 2003; Tseng et al., 2005; Jinsong et al., 2005). However, since these graphical notations are not as rich as e.g. UML class diagrams, seem to be less user-friendly, are less of an industrial standard, and seem to be tested very little, the presentation of such graphic notations, compared to using existing alternatives, cannot be justified. For this reason, such literature is not applied in this PhD thesis.

Finally, two other Danish research groups that have also carried out configuration research can be mentioned: Department of Production at Aalborg University (mainly K.A. Jørgensen) and Department of Mechanical Engineering at the Technical University of Denmark (mainly N.H. Mortensen). This work is most often presented in the form of conference papers. When used, they are referred to in the appended papers of this thesis.

2.3 Discussion of existing research

As a basis for the discussion of the relevance of the investigated literature, the model from chapter 1.3 (focus of thesis) is shown again in figure 4.

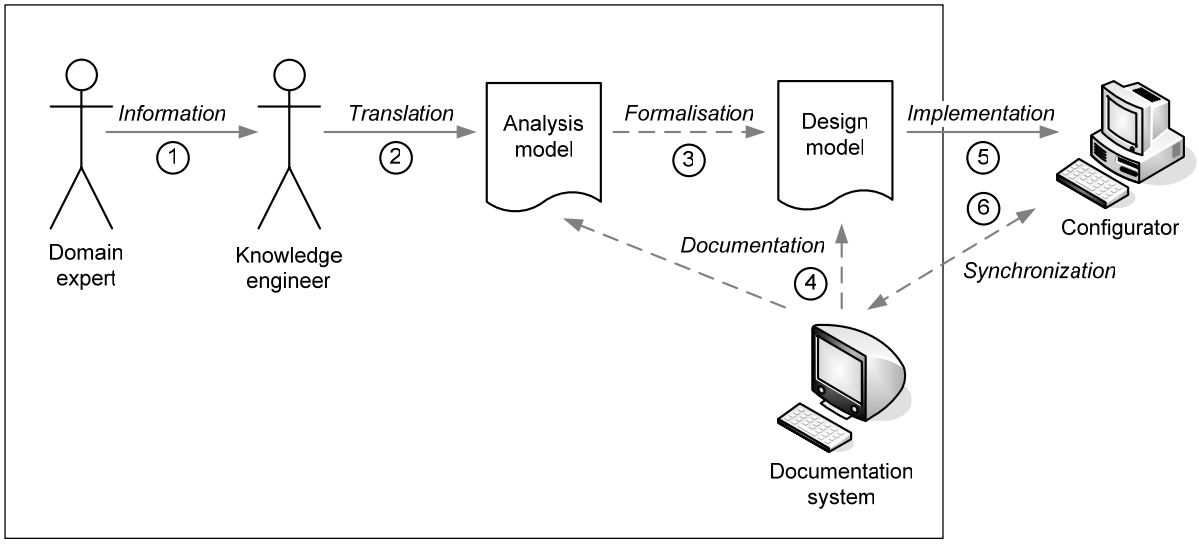


Figure 4: Focus of thesis

The contributions from Klagenfurt University focus mainly on defining methods that allow automation of the transfer of conceptual models to the knowledge base of configuration software, i.e. process 5 of figure 4. A relevant contribution from Klagenfurt University in relation to this PhD project is that they show how the UML can be utilized in configuration projects. By using UML concepts of stereotypes and profiles, UML is extended to reflect a configuration domain. Having such well defined syntax allows mapping to a configuration knowledge base, and thereby, automatic transfer of information. The focus of this PhD project is mostly on how to represent domain knowledge in a user-friendly and adequately expressive manner, and not as Felfernig et al., to make a language that automatically can be transferred into a configuration knowledge base. However, the representation techniques and methods of this PhD project could very well be integrated with the concepts of Felfernig to allow the extension of automatically importing conceptual models into the knowledge base of a configurator. Particularly in the context of creating a documentation system that supports the CPM-procedure, such ideas seem very relevant.

The Helsinki University of Technology has dealt with configuration mainly from a software technical aspect. Therefore, they also mainly deal with process 5 of figure 4. From the point of view of this thesis, the literature from the Finnish researchers mainly seems interesting with regard to defining the relevant concepts for a configuration context, i.e. what they term 'product configuration ontology'. Their research on knowledge representation focuses on defining formal representation formalisms that can be interpreted by computers, rather than user-friendly graphical languages to be used for knowledge acquisition, and is therefore not directly useful for this PhD project. However, their research in this area may be inspirational when dealing with defining formalisms for the expression of rules/constraints in graphical models.

The papers and the book by Forza and Salvador provide many insights into configuration projects through detailed analysis and by suggesting explanation models. But as mentioned, their descriptions of the knowledge acquisition process do not include detailed descriptions of representation techniques or reflections about the choice of representation techniques. This makes it hard to build only on Forza and Salvador in configuration projects of some complexity. Therefore, the contributions of this thesis could very well be seen as a needed extension of their work, by moving from a more principle level to a more concrete level on this point. The extensiveness of their book, while not describing the mentioned issues in detail, can also be seen as an illustration of the need for the research of the current PhD project, namely understanding the knowledge acquisition phase and modelling techniques that are used to describe product knowledge in configuration projects.

Compared to Klagenfurt University and Helsinki University of Technology, CPM has great focus on the first four processes of figure 4, while much less on process 5. The importance of being able to create adequately expressive, extensive and comprehensible representations is much emphasized in CPM literature. Therefore, it may seem strange that only CPM, of the four research groups in focus, goes to a great extent into the details of the first four phases of figure 4. Based on case studies and claims in the literature, it seems hard to attribute the neglect of this area to its being of less interest, on the contrary. A possible explanation for avoiding going into detail regarding the knowledge acquisition process of configuration projects may be its complexity. Dealing with the transformation of formalized models in one language to another formalized language does not involve the same unpredictable factors as in the knowledge acquisition process, which involves dealing with human beings who sometimes make irrational decisions and have different prerequisites, learning abilities, motivation etc.

In relation to process 1 of figure 4, research on the knowledge that domain experts possess, and the information they deliver, is almost untouched within configuration literature, although such understandings could have very positive impact on the way relevant tasks in configuration projects are perceived, and could provide an important basis for research on the knowledge acquisition process of configuration projects. In CPM literature and some other literature, the problems of retrieving the relevant information from domain experts are most often described in terms of tacit and explicit knowledge. The obvious question the application of this distinction gives rise to is, whether this division is suitable for explaining such phenomena and, in line with the purpose of this PhD, whether a better description can be made.

For process 2 of figure 4, the CPM-procedure prescribes the use of PVMs, based on claims of having experienced great success with the application of this technique for this purpose. However, detailed analysis of the use of PVMs in configuration projects has not been made. Therefore, possible limitations of the PVM technique are not described, except for it being mentioned that the PVM technique has limited richness and formality, without much further detail. Therefore, the obvious response to this issue is to investigate the use of PVMs in configuration projects and, in line with the purpose of this thesis, to suggest possible improvements of the formalism.

In relation to process 3 literature, the CPM-procedure prescribes the use of PVMs for (product) analysis models and class diagrams for (object oriented) design models. This prescription is based on the assumption that PVMs are easier to learn, whereas class diagrams are richer and more formal. However, CPM literature describes that PVMs and class diagrams have not both been used in all the projects in which the CPM-procedure has been applied. Often, a configurator has been built based on PVM models, while class diagrams have not been elaborated. The choice of diagrams depends on aspects such as prerequisites of domain experts, complexity of the product, the modelling environment of the standard configurator shell (or programming language) etc. However, since no more in-depth

studies of the CPM assumption concerning the basic characteristics of PVMs and class diagram have been made, it seems difficult to make a well-founded choice of representation techniques for a specific configuration project. It therefore seems of great relevance to investigate what the actual differences are between the application of PVMs and class diagrams in configuration projects. Providing such clarity about the actual differences of applying these two techniques raises another question, namely if a diagram that holds the advantages of both representation techniques can be defined. Applying such a diagram would imply that the prescribed time-consuming and error-risking task of transferring model information between PVMs and class diagrams could be avoided. For process 3, the issue of more advanced modelling concepts may also be necessary to deal with in many cases. There seems to be a need to extend the CPM literature with instructions of how to deal with such modelling aspects as interrelated models, class interfaces, other relationship types etc. In this context, a major issue is how to deal with separate models with overlapping content. This issue is relevant in contexts where separate analysis and design models are maintained, but maybe even more in cases where different model views on a product are needed.

For process 4 of figure 4, an effort is also clearly needed. In spite of the emphasis on the great potential benefits of having a documentation system that supports the modelling techniques of the CPM-procedure, such a system does not exist. In CPM literature, this topic has been investigated, but this has only resulted in superficial software specifications and not the required basis for the creation of such a documentation system. An important problem is that adequately clear and extensive definitions of the included notations and their mutual mappings do not exist. Probably mostly because of this shortcoming, previous attempts to create documentation system prototypes have only resulted in software with very little fulfilment of the requirements. Investigating which definitions are needed for a configuration documentation system to be created is therefore an important task.

Chapter 3: Research questions

Based on the analysis and discussion of configuration literature in the preceding chapter, formalized research questions can be defined. In doing so, I recognize the warning of Silverman (2005) that the aim should be to say "a lot about a little (problem)" and avoid the temptation to say "a little about a lot (of problems)". This basic insight is attempted incorporated in each of the research questions. Based on this, the research questions for this PhD thesis are formulated as follows:

1) Does the use of the term 'tacit knowledge' in configuration literature comply with the original meaning of the term, and does it make sense to apply this term in configuration research?

2) Can the knowledge/information that a domain expert possesses and delivers to a knowledge engineer in a configuration project be categorized in a better way than the tacit-explicit knowledge distinction?

3) What are the limitations of applying the PVM formalism, and how can the formalism be altered in order to solve such limitations?

4) What are the actual differences between using PVMs and class diagrams for modelling problems in configuration projects?

5) How can the migration of information from PVMs to class diagrams be avoided while not losing the benefits of the application of both techniques?

6) How can models with overlapping information in configuration projects be maintained, while avoiding the necessity to update the same information in several places?

7) What are the necessary definitions for the creation of a documentation system that supports the CPM-procedure?

Chapter 4: Frame of reference

Several terms and concepts found in the configuration literature, which are of importance for this PhD thesis, carry ambiguous meanings. This section therefore aims to clarify the definitions of the most important terms and concepts that I apply. Some theoretical elements that form a central basis for much of the research carried out in relation to the thesis are also summarized.

4.1 Configurators

To start with, two of the most central terms of this thesis are defined, namely 'configuration task' and 'configurator'. This section also briefly presents configurator reasoning principles and configurator application areas, in order to show the range of configurator technology.

4.1.1 The configuration task

Despite the growing interest in configuration research in recent years, there is not a single accepted definition of the central term, 'configuration'. However, there seems to be general agreement that configuration can be considered a special kind of design activity (Stumptner, 1997). Mittal and Frayman provide an often quoted definition of the configuration task:

> "Given: (A) a fixed, pre-defined set of components, where a component is described by a set of properties, ports for connecting it to other components, constraints at each port that describe the components that can be connected at that port, and other structural constraints; (B) some description of the desired configuration; and (C) possibly some criteria for making optimal selections.
> Build: One or more configurations that satisfy all the requirements, where a configuration is a set of components and a description of the connections between the components in the set, or, detect inconsistencies in the requirements." (Mittal and Frayman, 1989)

Sabin and Weigel (1998) describe the core of the configuration task as being to select and arrange combinations of parts that satisfy given specifications, and they emphasize that during this process, new components cannot be created or component interfaces be modified. They use the term 'component' to describe physical entities that are concrete parts that are available in the application domain. Sabin and Weigel provide a definition of 'configuration' as a type of design activity where:

> 1) "The artifact being configured is assembled from instances of a fixed set of well-defined component types"; and 2) "Components interact with each other in predefined ways." (Sabin and Weigel, 1998)

Soininen et al. provide the following definition:

> "Configuration as a task can be roughly defined as the problem of designing a product using a set of predefined components while taking into account a set of restrictions on how the components can be combined." (Soininen et al., 1998)

Felfernig at al. provide the definition:

> "A configuration task can be characterized through a set of components, a description of their properties, namely attributes and possible attribute values, connection points (ports), and constraints on legal configurations. Given some customer requirements, the result of computing a configuration is a set of components, corresponding attribute valuations, and connections satisfying all constraints and customer requirements." (Felfernig at al., 2000a)

Many other definitions exist, but often more vaguely defined than those mentioned. In the definitions by Mittal and Frayman, Sabin and Weigel and Soininen et al., it is stated that configuration tasks include a set of pre-/well-defined components. However, such definitions do not explicitly tell if the properties of these components can vary. On the other hand, the definition by Felfernig et al. does this explicitly and is therefore clearer on this point. Also, the four definitions seem to only focus on physical products.[8] Since it is also reasonable to talk about configuration of non-physical products, it could be better to use a term such as 'entity' instead of 'component'. Therefore, a definition that more clearly expresses the understanding of the term 'configuration' as it is applied in this thesis could be formulated as follows:

> "To configure means to combine predefined entities (physical or non-physical) and define their variable properties, while obeying constraints and legal interface combinations, in a way that satisfies given requirements."

4.1.2 Configurators and configuration systems

Having arrived at a definition of the configuration task, the next step is to look at the software tool that is often used to support the configuration task. From a software development point of view, configurators can be divided into three basic types: stand-alone software shells, ERP-system modules, and company-specific developed software. In the literature, two terms are often used to define this type of software, namely 'configurator' and 'configuration system'. While there seems to be agreement that a 'configurator' is a software system, the term 'configuration system' has been attributed two different meanings. When talking about the relevant type of software, Soininen et al. (1998) solely apply the term 'configurator'; Sabin and Weigel (1998) only apply the term 'configuration system'; while Felfernig et al. (2004) and Hvam et al. (2007b) use the terms 'configuration system' and 'configurator' interchangeably. Using the two terms interchangeably would not represent a problem if it were not for the fact that one of these terms is also used in reference to another thing. In this connection, Forza and Salvador apply the meaning of the term 'configuration system':

> "Configuration system: the set of human and computing resources that contribute to accomplishing the configuration and modelling processes" (Forza and Salvador, 2007).

Therefore, in the terminology of Forza and Salvador, a configuration system is more than the software application; it is also the humans around the configurator. In the papers that comprise this thesis, the terms 'configurator' and 'configuration systems' are used interchangeably about the specific software and not the socio-technical system that Forza and Salvador refer to. This is mainly due to the fact that the main theoretical basis originates from CPM and therefore builds on this terminology. However, to avoid misunderstandings, it may be better in the future to only apply the term 'configurator' to the specific type of software tool.

In the literature, configurators are defined as being expert systems and knowledge-based systems (KBSs). The term 'expert system' is often used interchangeably with 'knowledge-based system', although many argue that expert systems are merely a subgroup of KBSs (e.g. Jackson, 1999). In this thesis, the term (software) 'expert system' is applied to emphasize that a configurator performs actions that normally require human expertise. Jackson (1999) defines an expert system as "a computer program that represents and reasons with knowledge of some specialist subject with a view to solving problems or giving advice". In figure 5, the basic structure of an expert system is shown. The knowledge base contains the knowledge about the relevant domain, which the inference engine uses to draw conclusions.

[8] The ISO standards define a product as being both goods and services.

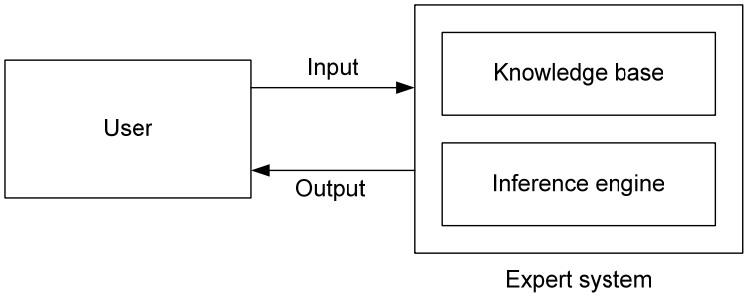


Figure 5: Basic structure of an expert system (based on Giarratano and Riley, 2005)

Based on the proposed definition of the term 'configuration' a definition of the term 'product configurator' could be formulated as follows:

> "A product configurator is a software-based expert system that supports the user in the creation of product specifications by restricting how predefined entities (physical or non-physical) and their properties (fixed or variable) may be combined."

This definition states that a configurator is a tool that supports the user when designing a product by making sure that the user avoids illegal combinations, not only of entities/components but also properties, including functions. Therefore, a configurator should not be confused with software tools that are capable of combining components, but which do so without any restrictions of rules or constraints. In my terminology, such tools can merely be defined as 'product modellers' or 'component selectors'.

4.1.3 Configurator problem solving

In an expert system, a programmer can express information about a problem to be solved in a declarative or a procedural manner (Giarratano and Riley, 2005; Hopgood, 2000). Declarative programs include facts, rules and relations, but do not state when this information should be applied, whereas procedural programs include a list of sequences to be carried out (Hopgood, 2000). Besides this basic distinction, other classifications exist with different division criteria of how configurators solve configuration problems.

Sabin and Weigel (1998) provide some examples of how configuration knowledge can be represented in a configurator:

- Rule-based reasoning
- Model-based reasoning
- Case-based reasoning

Rule-based systems use production rules as a uniform mechanism for representing both domain knowledge and control strategy (Sabin and Weigel, 1998). A (production or business) rule has the basic form: IF <condition> THEN <consequence>. Rule-based systems do not have a separation between their domain knowledge (represented by directed relationships) and their control strategy (represented by procedural knowledge that controls the computation of a solution). This lack of separation and the spread of knowledge of a single entity over several rules can make maintenance extremely difficult (Sabin and Weigel, 1998). For instance, when making changes, it is very hard to be certain that all the rules that need changes have been found, and if a condition does not trigger, the system does not apply the 'consequence' part of a rule.

Model-based reasoning builds on the presence of a systems model, which contains decomposable entities and interactions between their elements. Compared to rule-based reasoning, model-based reasoning allows a separation between domain knowledge and how this knowledge shall be used. It has enhanced robustness (ability to solve a broader range of problems); enhanced compositionality (better ability to combine knowledge from different domains within a model); and enhanced

reusability (better ability to use existing knowledge to solve related types of problems) (Sabin and Weigel, 1998). According to Sabin and Weigel (1998), the most relevant model-based approaches are logic-based (often based on description logics), resource-based (entities produce or consume resources), and constraint-based reasoning (restrictions on how elements and their properties may be combined).

Case-based reasoning provides another way of reasoning. It does not use either deductive or abductive schemes to derive conclusions, but uses previously solved problems that are adapted to the current requirements. The basic reasoning cycle consists of the following: 1) retrieve most similar cases; 2) reuse information of the cases to solve the problem; 3) revise the proposed solution; 4) retain the parts of the experience likely to be useful for future problem solving (Aamodt and Plaza, 1994).

Stumptner (1997) provides an (admittedly incomplete) overview of the state of configuration research, mentioning three categories of approaches:

- Representation-oriented methods
- Task-oriented approaches
- Hybrid systems

In brief, representation-oriented approaches are described as focusing on finding an efficient representation in a configurator for describing the problem domain. Representation-oriented approaches include among others: rule-based configuration; structure-based configuration; constraint-based configuration; and resource-based configuration. Task-oriented approaches focus on defining the sub-problems that are to be solved. The idea is that different aspects of a problem are best solved as different kinds of reasoning tasks with overall strategic decisions that influence which task is attacked at a particular point in the problem-solving process. Hybrid systems exhibit aspects of the first two approaches. This approach includes case-based reasoning, among others.

According to Soininen at al. (1998), earlier conceptualizations of configuration knowledge can be roughly classified as:

- Connection-based (Mittal and Frayman, 1989)
- Resource-based (Heinrich and Jüngst, 1991)
- Structure-based (Cunis et al., 1989)
- Function-based (Najman and Stein, 1992)

According to Soininen at al. (1998), these four approaches have little in common except the central notion of a component. But the experiences of Soininen at al. (i.e. Tiihonen, 1994; Tiihonen, et al., 1996) have led them to believe that the four types of modelling concepts are all needed to compactly and adequately represent the knowledge on products.

In this thesis, the focus is on what Sabin and Weigel (1998) define as model-based reasoning and what Stumptner (1997) defines as representation-oriented methods. With this focus, in principle, the four types of conceptualizations of configuration knowledge defined by Soininen at al. (1998) are also supported by the representation formalisms proposed in this thesis.

4.1.4 Configurator application areas

Forza and Salvador define four types of product customization with relevance for distinguishing between different types of applications of product configuration: customized distribution, customized assembly, customized fabrication, and pure customization (customization in the design phase). This is illustrated in figure 6.

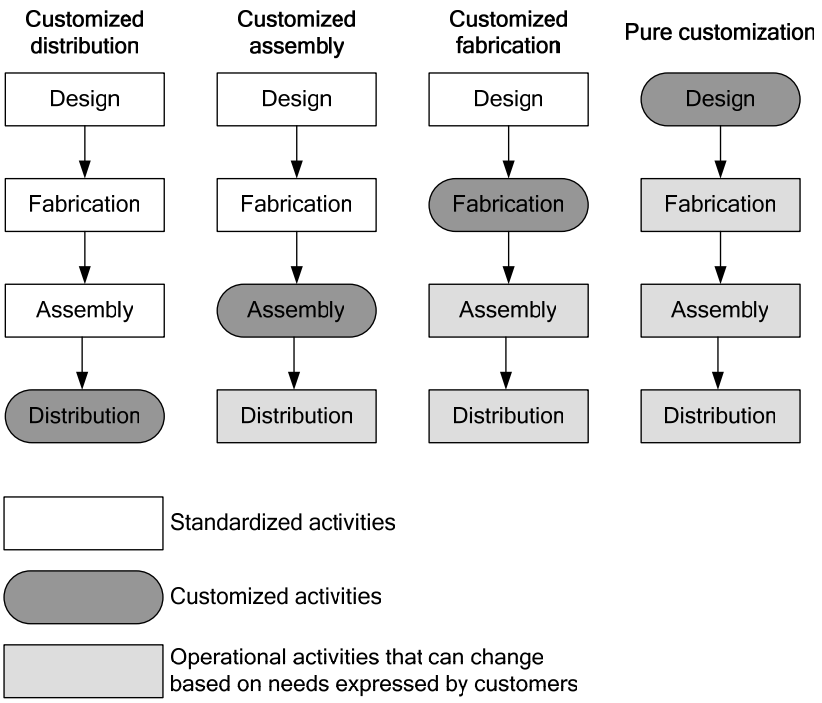


Figure 6: Configuration scope (simplified version of Forza and Salvador, 2007)

Forza and Salvador (2007) distinguish between commercial and technical configuration systems. They define the commercial configuration process as "all the activities carried out to identify the complete and congruent commercial description of the product that best fits customer requirements", and the technical configuration process as "all the activities that generate the documentation of the product variant based on the commercial description of such variant".

4.2 Models

This section resumes some of the model classifications that are found in the configuration literature. First, traditional software analysis and design models are described. Next various classifications of product models and configuration models are described. The classifications resumed are not analysed or discussed, but are included to illustrate and define the multitude of possible models in configuration projects and to underline the relevance of being able to maintain models with different views of the same product efficiently. This section is therefore most explicitly important for research question 6 concerning maintenance of models with overlapping content.

4.2.1 Software models

In software development, the two most common types of models are analysis models and design models. Analysis models are created early in a software project and can help to establish an understanding of the area that is addressed by the system before the activities of system design and coding are initiated. Analysis models do not directly refer to the properties of the software system, but focus more on investigating reality than creating a solution (Priestley, 2003; Larman, 2002). Analysis models also go under the names of 'conceptual models' and 'domain models'.

Design models describe the architecture of a program at a higher level of abstraction than source code, and can be an invaluable help for software engineers to analyse program architectures, design choices, behaviours, and implementations (Guéhéneuc and Albin-Amiot, 2004). Normally, a design model contains much of the same information as analysis models, but includes more details and makes references to properties of the proposed software system (Priestley, 2003; Larman, 2002).

In configuration projects, analysis models are often elaborated in order to collect and document relevant product knowledge with few considerations of how this should later be implemented into the knowledge base of a configurator. In configuration projects, analysis models are also used as a basis for discussions of what is to be included in the project; i.e. which parts of the product assortment are suited for configuration; which parts of the product should be standardized etc. Thus, the produced analysis models are often not suited for direct implementation in configurator software and may have to be redesigned in order to be supported by the chosen software. Also, in some cases, the analysis models can be simplified without losing information, which can later ease the implementation and maintenance of the configurator. Therefore, the application of design models allows a more fluent transition from paper model to implemented model in the configurator knowledge base. Another aspect is that in some projects only parts of what is described in the analysis model are to be implemented at a specific stage. For this reason, design models can be used to define this selection, while the remainder of the contents of the analysis model is saved for possible later implementation.

Capturing the relevant aspects of the real world in analysis models in a way that enables discussions between relevant parties can be very challenging; therefore, the inclusion of implementation issues at this stage would involve additional complexity. Thus, the distinction between analysis and design models can be very useful when developing configurators. But this differentiation between analysis and design models does not mean that there necessarily have to be big differences between such models; in many configuration projects, the design models are merely refined or extended versions of the analysis models, without any real demarcation of the move from one model to the other.

When a design model has been implemented and the configurator is taken into use, the project moves into the maintenance phase. In the maintenance phase, design models can be used as external documentation of what is implemented. However, to have useful external documentation, this has to be updated when changes occur. If only the design models are updated when changes are made to the knowledge base of the configurator, a discrepancy between the implemented knowledge and the analysis model emerges. This could be problematic in cases where changes of the configurator knowledge base require the involvement of domains experts who are only familiar with the analysis language. On the other hand, to maintain both analysis and design models can be very time consuming, and it can be difficult to keep the models consistent, especially if they are created using different modelling techniques, such as proposed by the CPM-procedure. The creation of a software tool that supports CPM-procedure in a way that allows integrated maintenance of such models could change this, but currently such a system does not exist. This issue is targeted in relation to answering research question 7 of this thesis.

4.2.2 Product models

The terms 'product model' and 'product modelling' frequently appear in the configuration literature. In general, the term 'product model' can refer to a lot of different kinds of models, such as physical models, hand-drawings, 2D/3D CAD models, textual specifications etc. However, in the context of developing product configurators, one type of product models is particularly interesting, namely the models that describe the product information that eventually ends up in the knowledge base of the product configurator. This kind of model could more precisely be named 'generic product information model'. Later, when a configurator is taken into use, instance models are produced, such as BOM's, 3D drawings, 2D drawings etc.

To provide an overview of relevant product models in a configuration context, Hvam et al. (2007a) present a framework for structuring product knowledge in product configurators, as shown in figure 7. The framework describes product models in terms of their properties, functions, organs and parts, based on the 'theory of domains' described by Andreasen (1992; 1998). In addition, the framework includes models of a product’s meeting with life-phase systems.

	Property models (Derived properties)		Product structre model		Models of the product's meeting with life cycle systems			
	Internal and external properties Describes consequences of meeting between product and life cycle systems	**Functional properties** Describes product's function	**Solution principles** Describes product's function-bearing units	**Part model** Describes product's components	**Factory model** Overall description of production equipment, layout etc.	**Process model** Detailed description of individual manufacturing processes and production equipment	**Assembly model** Describes assembly of product	**Transport model** Describes transport of product	**Other life cycle models** Can for example be service or disposal / recycling
Generic level	Rules for calculating internal properties, e.g. fatigue strength. Rules for calculating external properties, e.g. lifetime.	Rules for describing the product's function and its relation to the function-bearing units (solutions in principle)	Rules for describing solutions in principle and their relation to functions and parts	Rules for describing parts and their relation to solutions in principle	Rules for selection of production equipment and rules for calculating time consumption etc.	Rules for describing the individual production processes	Rules for selecting assembly equipment and calculating assembly time	Rules for selecting form of transport and calculating transport price	
Instance level	e.g. Ultimate strength Product life time Cost price etc.	Functional description	Description / definition of solutions in principle	Drawing, list of parts etc.	List of operations, production layout, description of production equipment etc.	Process description, layout description, description of tools, CNC code etc.	Assembly instructions, list of assembly equipment, assembly time etc.	Transport price, description of packaging, transport documents, etc.	

Figure 7: Framework for structuring product knowledge in product configurators (Hvam et al., 2007b)

4.2.3 Configuration models

Soininen et al. (1998) defines three classes of configuration knowledge:

- Configuration model knowledge
- Configuration solution knowledge
- Requirements knowledge

Configuration model knowledge includes definitions of the entities that can appear in a configuration, their properties, and the rules for how entities and their properties may be combined, i.e. the information needed to define a product based on given requirements. Models of configuration solution knowledge specify a specific configuration or partial configuration. Models of requirements knowledge show the requirements for the configuration that must be constructed. The three classes of knowledge correspond to what is used in the three phases of a configuration task, namely providing input (requirements), configuration processing (configuration model knowledge), and delivery of output (configuration solution knowledge).

Based on Harlou (2006), Hvam et al. (2007a) operate with three levels of product views that can be relevant when developing product configurators: customer view; engineering view (development); and part view (production).

Forza and Salvador (2007) define two types of generic product data models: a commercial model as "a formal representation of the product space and of the procedures according to which a commercial configuration can be defined within such space", and a technical model as "a formal representation of the links between commercial characteristics and the documents that describe each product variant (bills of material, production and assembly cycles etc.)".

4.3 Knowledge representation

The term 'knowledge representation' is central in this thesis. Therefore, this section defines what knowledge representation is, which representation techniques are applied in a configuration context, and which modelling constructs are needed to represent configuration problems.

4.3.1 Knowledge representation principles

Knowledge representation is a subfield of artificial intelligence. Brachman and Levesque provide the following definition of knowledge representation:

> "...the field of study concerned with using formal symbols to represent a collection of propositions believed by some putative agent." (Brachman and Levesque, 2004)

The use of knowledge representations is a central element when developing expert systems, including configurators. Knowledge representations can help people to visualize knowledge and maintain and develop a common understanding before implementation (Gordon, 2000).

The development of KBSs was in the early 1980s seen as a transfer process of human knowledge into a KBS knowledge base. This idea is based on the assumption that the knowledge that is required by the KBS already exists and just has to be collected and implemented (Studer et al., 1998). Today, however, there exists a general consensus that the development of a KBS should be perceived as a modelling activity (Studer et al., 1998, Speel et al., 2001). Thus, building a KBS does not aim at realizing problem-solving capabilities comparable to a domain expert, but instead at creating a model which offers similar results in problem-solving. Studer at al. (1998) argue that the modelling view of the process of building a KBS of today has the following consequences:

- A model is only an approximation of the reality, and in principle, the modelling process is infinite, because it is an incessant activity with the aim of approximating the intended behaviour.

- The modelling process is cyclic, where new observations may lead to a refinement or modification of the already built-up model.
- The modelling process is dependent on the subjective interpretations of the knowledge engineer, wherefore this process is typically faulty. An evaluation of the model with respect to reality is indispensable in order to create an adequate model.

Cawsey (1998) presents five general requirements for a representation language:

- Representational adequacy: allow representation of all knowledge needed to reason
- Inferential adequacy: allow new knowledge to be inferred from a basic set of facts
- Inferential efficiency: how quickly inference can be done
- Clear syntax and semantics: clear definitions of valid expressions
- Naturalness: reasonably natural and easy to use

Jackson (1999) provides a definition of the main criteria for a knowledge representation:

- Logical adequacy: is the representation capable of making the needed distinctions?
- Heuristic power: does the representation allow drawing the required inferences and solving problems within its intended domains of application?
- Notational convenience: can the language can be read, written, and understood?

4.3.2 Modelling techniques

As mentioned, representation of the relevant domain knowledge is often described in the literature as one of the major challenges in a configuration project. While the researchers from University of Klagenfurt and Helsinki University of Technology apply class diagrams for the creation of graphical models, the literature from CPM includes three techniques that are used for describing domain knowledge and designing the knowledge base of a configurator, namely:

- Product Variant Masters (PVMs)
- Class diagrams
- CRC-cards (extended version)

The PVM technique (originally named Product Family Master Plan (Mortensen, 1999)) is a graphical notation for describing product families. A PVM model consists of two generic sections, part-of and kind-of structures. The part-of section describes the parts of which a given product family can consist, while the kind-of section describes the variations of a given part, i.e. different types with common characteristics. Since the PVM technique was included in the CPM-procedure, the formalism has been subjected to some changes. In figure 8 and 9 the notation formalism of the PVM technique as it has been defined until recently and the latest definition of the formalism are shown. The main changes of the PVM formalism are: the term 'class' is used instead of 'module', 'unit' and 'part'; kind-of classes are lined up vertically, whereas previously they were lined up horizontally; constraints are no longer shown by drawing lines between classes, but are written below a class; cardinality is shown; and descriptions of classes are included in the formalism.

Due to the time of the appearance of the revised version of the PVM formalism, some of the papers included in this thesis build on the older definition.

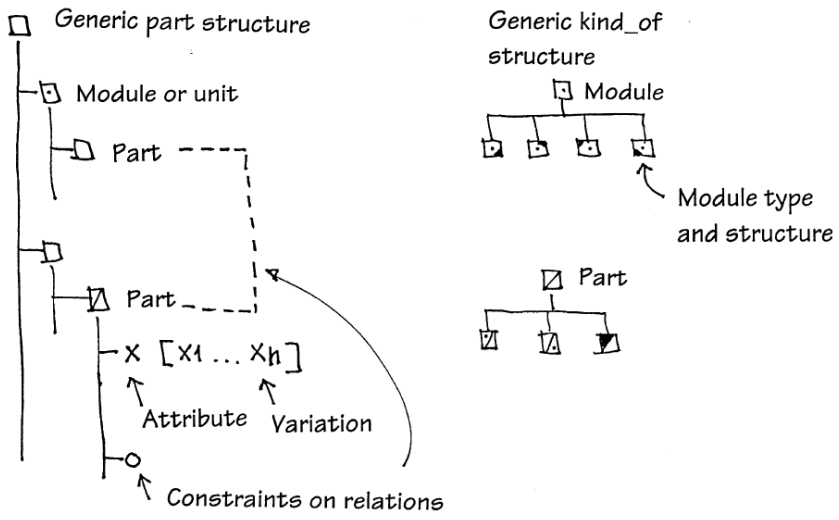


Figure 8: PVM formalism (Mortensen et al., 2000)

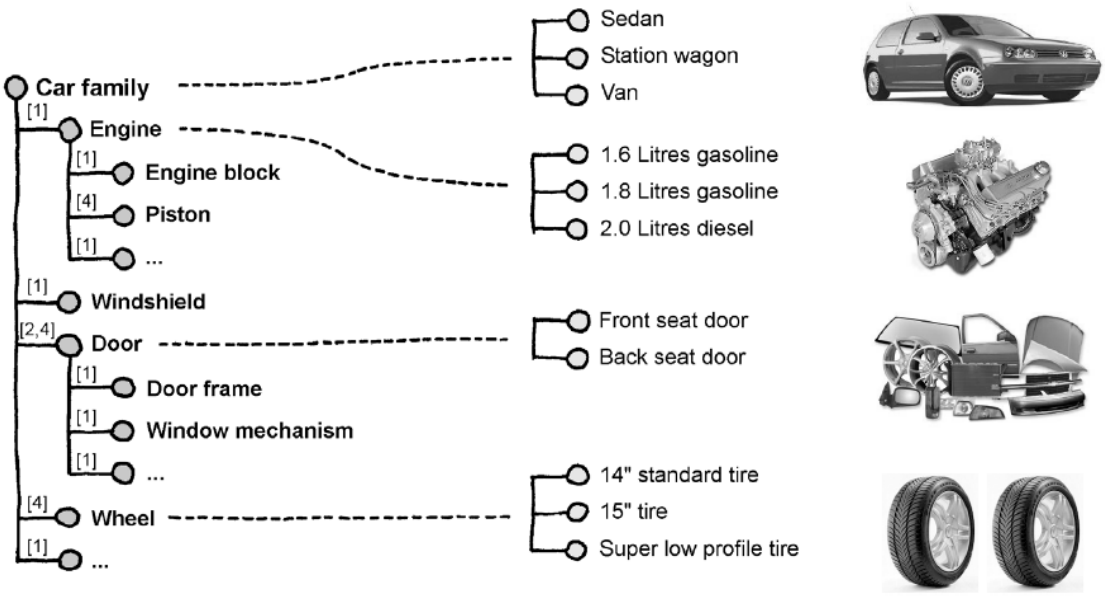


Figure 9: PVM formalism (Hvam et al., 2007b from Harlou, 2006)

Class diagrams are far better known internationally and more widely used than PVMs. Class diagrams are part of the Unified Modelling Language (UML), which officially became a standard of the Object Management Group (OMG) in 1997. This PhD thesis is based on the 2005 specification of UML 2.0 (OMG, 2005). UML 2.0 includes thirteen diagram types, which are divided into three categories: six structure diagrams (including class diagrams), three behaviour diagrams, and four interaction diagrams (subtype of behaviour diagrams).

Class diagrams are used for depicting the static structure of a system by describing the objects and the relationships between them. A class is a set of objects that share common features. In UML terminology the term 'features' refers to the 'properties' and 'operations' of a class. The properties are the structural features that appear in two distinct notations, 'attributes' and 'associations', which are more or less used to express the same thing (Fowler, 2005). Operations are the actions that a class knows how to execute, which therefore determine the behaviour of a class. Often the terms 'operation' and 'method' are used interchangeably, but in fact there is a difference in that operations describe the

processes performed by a class, while a method as an implementation of an operation, i.e. the code to be executed.

The notation for the class element and the most common relationship types is shown in figure 10, where a navigability arrow can be used to show the direction of association, aggregation and composition relationships.

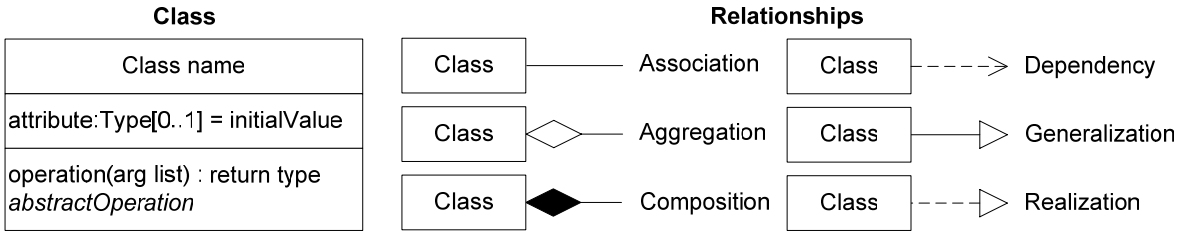


Figure 10: Basic elements of UML class diagrams

A useful concept in UML is stereotypes. Stereotypes represent variations of existing types of model elements (e.g. classes or relations). Hereby, the use of platform or domain specific terminology or notation is possible. All UML model elements can be extended by a stereotype, and the name of a stereotype is placed in guillemets (<<...>>). Together with tagged values and constraints, stereotypes can be collected in profiles. Profiles were added to the UML 1.4 specification and are packages that contain customized model elements for a specific purpose (OMG, 2005).

Another element of UML that is sometimes useful is the formal constraint language 'Object Constraint Language' (OCL). However, constraints can also be formulated within normative UML without the use of OCL, since any formalism used for describing constraints is allowed, as long as it is placed within braces ({}) (Fowler, 2005).

A UML diagram type that is also very relevant as an extension of class diagrams is package diagrams. A package is a construct for grouping UML elements (most often classes) together in higher level units. This allows organization of models so that they are easier to overview. In figure 11, package diagram notation is displayed.

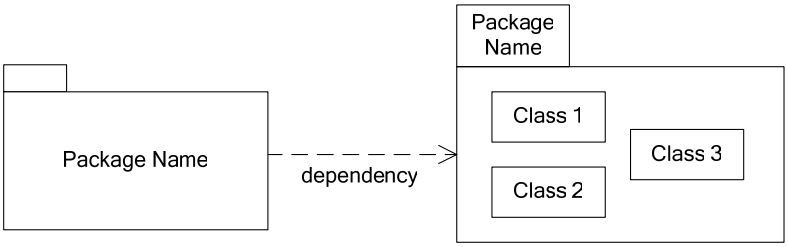


Figure 11: Package notation (based on Fowler, 2005)

The last of the three main modelling techniques of the CPM-procedure is CRC-cards. CRC-cards were invented by Ward Cunningham in the late 1980s (Fowler, 2005) and originally presented in a paper by Beck and Cunningham (1989), where they are described as a way of teaching the object-oriented way of thought. The original CRC-card consists of the 'class name' together with two columns for 'responsibilities' and 'collaborators'. In brief, responsibilities are summaries of the things that an object from the class should do, while collaborators are the other classes that the class works together with (Fowler, 2005).

In the CPM-procedure the purpose of CRC-cards is to organize detailed information about classes represented in PVMs and class diagram models. For this reason, the CRC-cards have been extended with additional fields compared to the original definition. Since the definition was presented in Hvam (1994), the CPM CRC-cards have evolved to the layout shown in figure 12.

Class name:	Date:	Author/version:
Responsibilities:		
Aggregation	Generalisation	
Superparts:	Superclasses:	
Subparts:	Subclasses:	
Sketch:		
Attributes:	Collaborators:	
System methods:		
Product methods:		
Internal methods: External methods:		

Figure 12: The latest CRC-card definition by CPM (trans. from Hvam et al. 2007a)

In the CRC-cards by CPM, the methods fields are for maintaining information about both methods and what in other contexts are referred to as constraints or rules. In the CRC-card, product methods concern products and their life-phase properties, while system methods concern software aspects of a configurator. Internal methods refer to the internal structure, functions and properties of a class, while external methods refer to interfaces to others classes (Hvam et al., 2007a).

UML class diagrams have proved to be a very popular modelling language for creation of knowledge representation within development of product configurators, and particularly in Danish projects, PVMs combined with CRC-cards have often been applied. But other graphical representation techniques are also found in expert system literature. However, these are mostly of older date (e.g. 'semantic nets' and 'frames') and are not much used in the type of configuration projects in focus in this thesis. For descriptions of such techniques, see e.g. Jackson (1999), Gordon (2000), Brachman and Levesque (2004) or Giarratano and Riley (2005).

4.3.3 Configuration modelling concepts

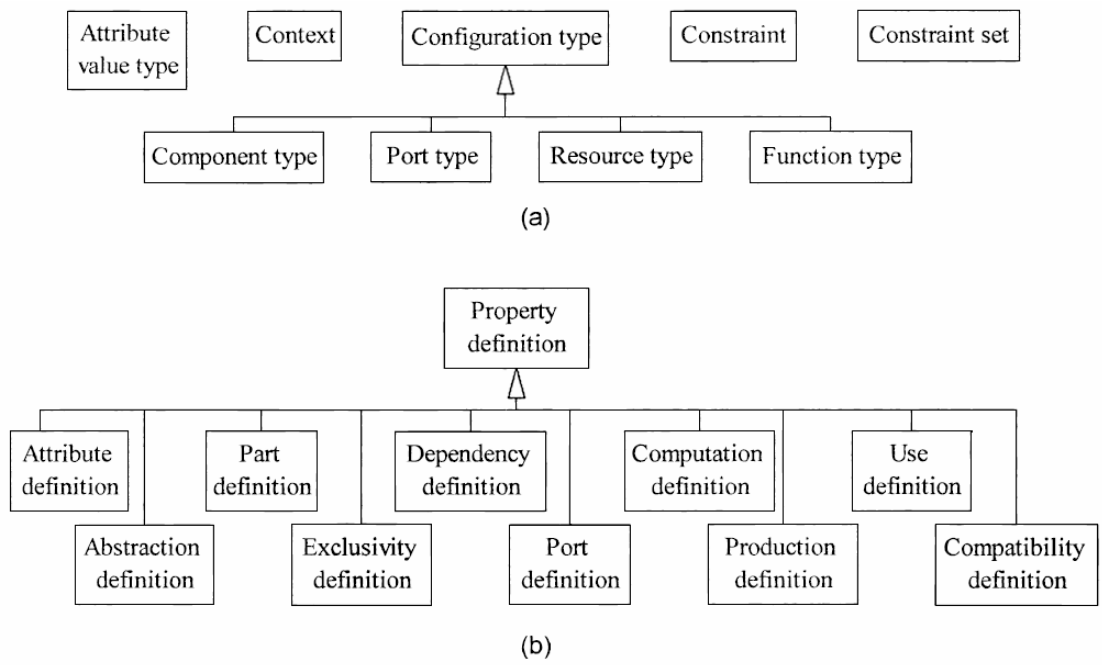
Some researchers have made efforts to define which model constructs are needed in order to represent a configuration problem. According to Sabin and Weigel (1998), any configuration framework must address the issues of: expressiveness and representational power; efficient knowledge application in a highly combinatorial context; and mechanisms for coping with the high rate at which knowledge changes. Application objects (~ classes) are unique in that they having individual properties that fall into three categories:

- Attributes (specify descriptive features of objects and have single values or a predefined range of values, discrete or continuous)
- Resources (specify (functional) characteristics that a system can supply or use)
- Ports (places through which objects are connected and communicate)

In addition, application objects have an internal structure that is described by a set of constituent objects and the interconnections between them. According to Sabin and Weigel (1998), a configurator should as a minimum be able to handle the following relationship types:

- Classification (kind-of)
- Aggregation (part-of)
- User-defined relations among sets of objects existing independently
- Local constraints (structural, arithmetic, geometric, cardinality etc.)
- Global constraints (resource constraints, optimization criteria etc.)

Soininen et al. (1998) present a generalized ontology of product configuration. The ontology consists of concepts for the representation of configuration knowledge, including restrictions on possible configurations. The aim of the ontology is to synthesize the conceptualizations of the connection-, resource-, structure- and function-based approaches. From the connection-based approach, they include the concepts 'component', 'port' and 'connection constraint'; from the resource-based approach, the concept of 'resources' that components produce and use is included; from product structure based approach, they include the concept of 'compositional structure'; and from the functional approach, they include the concept of 'functions' implemented by components. In figure 13, the classes and their taxonomy in the ontology is shown. Part (a) shows the main concepts, part (b) shows the property definitions that can be given to configuration types, and (c) shows further classes that are used in property definitions.



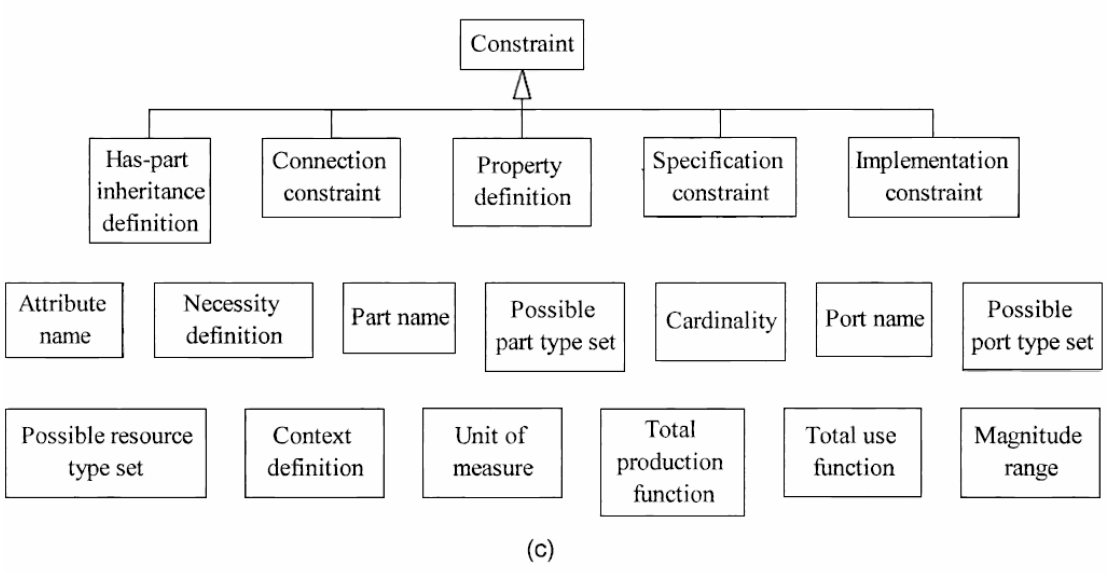


Figure 13: Overview of classes and their taxonomy (Soininen et al., 1998)

Based on the product configuration ontology of Soininen at al. (1998), Felfernig et al. (e.g. 2000a; 2000c; 2001) select some of the basic concepts and show how UML class diagrams can be used to model them. Felfernig et al. (2001) include:

- Component types (parts of which the final product can be built, characterized by attributes with predefined domains of possible values)
- Function types (for modelling the functional architecture, characterized by attributes)
- Resources (some of the component/function types are producers of resources and others are consumers)
- Generalization (inheritance from general component/function classes to specialized classes)
- Aggregation (part-of structures that state a range of how many subparts an aggregate can consist of)
- Connections and ports (for describing how components are interconnected with each other)
- Compatibility relations (for defining if components/functions are incompatible or require others)
- Constraints (constraints on the product model that cannot be expressed graphically are formulated using OCL)
- Additional modelling concepts (i.e. concepts not covered in their definition)

The means for extending UML to include the mentioned concepts is to define additional modelling concepts inside UML by introduction of a 'profile' (Felfernig et al., 2001). Felfernig et al. (2001) define the class stereotypes 'component', 'resource', 'function' and 'port', the specialized associations 'incompatible' and 'is_connected', and the specialized dependencies 'requires', 'produces' and 'consumes'. In figure 14, an example of the use of these definitions is shown.

Configuration knowledge models often include both a structural and a functional architecture. In Felfernig et al. (2001) these two views of a product are interrelated through a mapping between functions and physical components, which can be expressed through the dependency relationship or additional constraints. This is illustrated in figure 15.

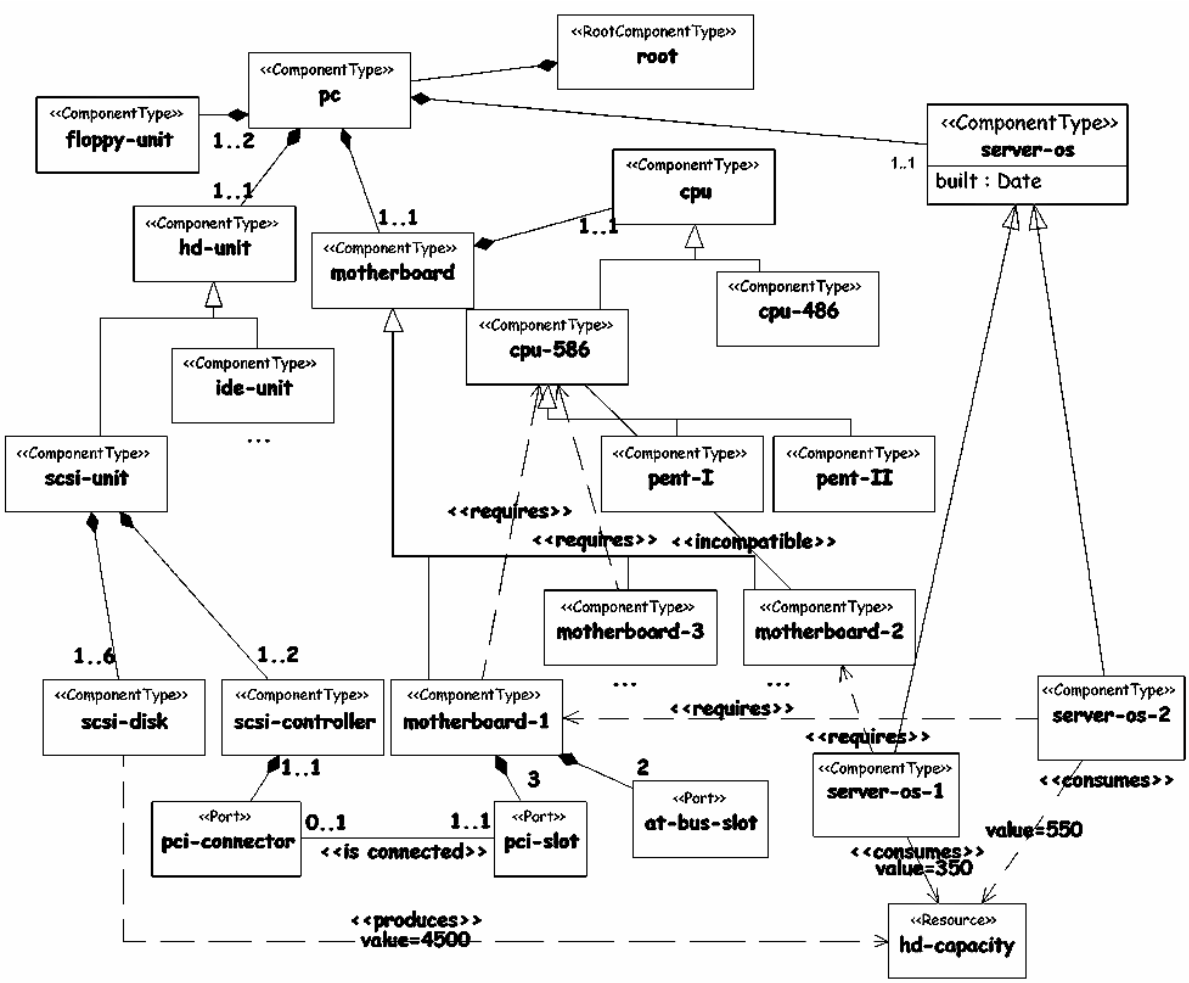


Figure 14: Conceptual model (Felfernig et al., 2001)

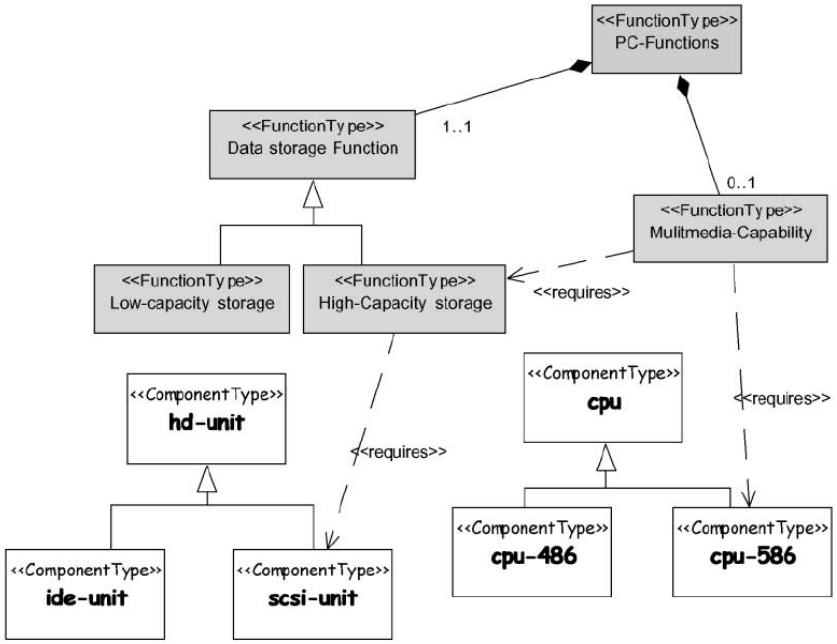


Figure 15: Functional product model (Felfernig et al., 2001)

Finally, Felfernig et al. (2001) mention the use of the structuring mechanisms: UML packages (to partition models and define model views) and contextual diagrams (Felfernig et al., 2000d).

A more basic set of constructs for describing configuration knowledge is presented by Hvam et al. (2007a). They recommend using the following subset of UML class diagram model elements:

- Classes (can be used to represent components, functions and properties)
- Attributes (name of attribute and value domain)
- Methods (also applied for expressing constraints/rules)
- Generalization (UML relationship type)
- Aggregation (UML relationship type)
- Association (UML relationship type)

In addition, Hvam et al. recommend the use of UML package diagrams for organizing models.

As mentioned, a basic purpose of this thesis is to build on and improve the methods and techniques of the CPM-procedure. On the basis of the prescription of the CPM-procedure that the process of representing product knowledge in configuration projects should allow inclusion of domain experts, the definition of constructs by Soininen et al. (1998) seems unnecessarily complex. It includes too many elements that are not clearly defined, but still does not seem to cover the needs in all cases. On the other hand, the selection of constructs by Felfernig et al. seems well-suited in this aspect.

4.4 The CPM-procedure

A central basis for much of this thesis is the CPM-procedure, since the procedure (or parts of it) has been applied in many of the cases investigated, and because a central goal of this thesis is to improve the knowledge representation techniques that are included in this procedure.

Since the emergence of the CPM-procedure in 1994 (Hvam, 1994), the CPM-procedure has been through much development. Its current form is shown in figure 16.

Phase		Activity	Tools
1	**Development of specification processes**	Step 1: Identification and characterization of the most important specification processes. Step 2: Formulation of aims and requirements for the individual specification processes. Measurement and gap analysis. Step 3: Design of new specification process. Definition of the configuration system(s) which are to support the specification process. Step 4: Evaluation and selection of scenarios. Step 5: Plan of action and organization of further work.	Flow charts, activity chains or IDEF0 Targeting and gap analysis Framework for structuring product knowledge Other tools: ● SWOT analysis ● Scenario techniques ● Cost-benefit analysis ● Benchmarking ● Use case diagrams ● Project management ● Change management
2	**Analysis of product assortment**	Analysis of product assortment. Definition of configuration system's overall content and structure. Design of product variant master.	Product variant master possibly associated with CRC-cards Framework for structuring product knowledge Other tools: ● Modularization ● Scenario techniques
3	**Object-oriented analysis**	Construction of object-oriented analysis (OOA) model.	Class diagram with associated CRC-cards and other UML diagrams
4	**Object-oriented design**	Choice of configuration software. Adaptation of OOA model to the chosen configuration software. Elaboration of requirements specification for programming, including user interface, integration with other systems and	Forms of knowledge representation Criteria for choice of software Class diagram with associated CRC-cards and other UML diagrams Other tools:

		programming dynamics.	● Other UML diagrams
5	**Programming**	Programming and test.	Configuration software
6	**Implementation**	Implementation of configuration system and the future specification process.	Plan for implementation Training of users of the system Other tools: ● Change management
7	**Maintenance**	Measuring and follow-up on the new specification process. Maintenance and continual further development of configuration system. Appointment of persons responsible for maintenance and further development.	Measurement methods Plan for organization of system maintenance

Figure 16: The procedure for building product models (trans. from Hvam et al., 2007a)

In spite of looking like a waterfall model of development, the idea is to apply iterative cycles through the procedure, which will result in repeatedly refined definitions.

1) Development of specification processes

The purpose of the first phase is to analyse the business processes of the specific company in order to clarify the business goals and define the future specification processes. The first step of the phase is to identify and to characterize the most important specification processes. This can be done by identifying the most important specifications, e.g. quotations, BOMs, list of operations, and the processes in which these are created. The specification processes are often described in flow charts to ensure a common understanding of their content. The next step is to formulate goals and requirements for the most important specification processes. To make the analysis concrete, measurements of various aspects of the specification processes can be carried out, such as lead times, use of resources, errors etc. Step three is to create scenarios for how the future specification processes can be defined, based on the use of configuration technology. This involves defining the scope of the configurator for the given scenarios. Next step is to evaluate and select among the defined scenarios by using a cost-benefit evaluation and related techniques. Finally, after having chosen a scenario, a budget and a plan of action are defined in step five. The latter includes descriptions of the tasks that need to be carried out, milestones, organization issues and more.

2) Analysis of product assortment

The purpose of the analysis of the product assortment is to create an adequate overview of the relevant products and describe the product knowledge that is later to be implemented into a configurator. The assumption is that domain experts should be able to participate in the work of creating the models of product knowledge. This obviously demands that the applied techniques are relatively comprehensible for persons without a thorough knowledge of IT or conceptual modelling languages. Therefore, in phase 2, PVMs together with CRC-cards are the main techniques for knowledge representation, while class diagrams are prescribed for later phases. The normal procedure of the acquisition process is to draw the PVM models on large sheets of paper, and refine the PVM models in sessions with knowledge engineers and domain experts. Normally, PVMs are drawn using software such as MS Visio, Excel and CAD-programs.

The work carried out in this phase can lead to redefinition of the product assortment, e.g. if some of the offered variants are not profitable or better solutions exist. Some of the issues discussed in this phase are: customer groups in focus; how the offered product variants match the relevant market; whether some products should be excluded or new included; most relevant product properties for customers; legal requirements for the relevant products; and services offered in connection with the products. To provide a basis for the subsequent phases, some more technical aspects should also be discussed, such as delimitation of product variants and marked segments; stability of the product assortment; complexity of the products; required level of detail in the configurator; localization of product knowledge; definition of the needed calculations for creating product variants; and whether these calculations should be included in the configurator.

3) Object-oriented analysis

The purpose of the object-oriented analysis phase is to further develop the models from the preceding phase into more formal models that in addition to product knowledge also include software related aspects. This process can also include simplifications of parts of the models in cases where classes of the PVM can be left out or merged to simplify the model that is to be implemented in a configurator. The models are formalized using a sub-selection of UML class diagrams together with CRC-cards. Besides these techniques, 'use case diagrams' (UML diagram type) can be used to define user requirements for the configurator and the integration with other IT systems.

Also, the user-friendliness of the user-interface of the configurator is to be considered. Here, the procedure is based on the points from Rogoll and Piller (2004): ease of operation; ease of navigation; individual access to information; waiting time; and support. Finally, specifications of system requirements, system dynamics and integrations with other systems are to be produced in this phase.

4) Object-oriented design

The purpose of the object-oriented design phase is to select software, adapt the object-oriented analysis models to the chosen software, and define requirements specifications for the programming phase. For selecting the configurator software, a list of possible requirements is suggested, e.g. response times, support possibilities, software structure etc. When adapting the object-oriented models, an important aspect is that much standard configurator software is not fully object-oriented, which can therefore require redefinition of parts of the models. Besides the adaptation task, a detailed user interface specification, definitions of dynamics, and integration with other systems are to be considered in relation to the selected software. Before initiation of the programming, requirements for the system are to be summarized and further specified. The purpose of this is to provide an adequately detailed picture of what is needed in the configurator. Some of the areas of requirements to be considered are related: user-friendliness; reliability; accessibility/security; ease of maintenance; performance aspects; system interfaces; requirements for software environment; and elaboration of system documentation.

5) Programming

On the basis of the produced specifications, the configurator is developed. This task differs depending on whether standard configurator software or merely a programming language is chosen. The procedure emphasizes the need for involving users in testing the configurator during the programming, both in order to educate users and to evaluate the system.

6) Implementation

The implementation phase focuses on the implementation of the configurator in the organization. The procedure suggests a number of steps for ensuring the users' accept of the system, such as: ensuring the system's user-friendliness; early user involvement in tests; providing clear information about consequences of the system; training users; and rewarding users for using the system.

7) Maintenance and further development

After having implemented the configurator in the organization, it must be maintained and further developed. Since modelling environments most often do not provide adequate overviews of the implemented information, external documentation can be necessary. According to the CPM-procedure it is often adequate to document the system using either PVMs together with CRC-cards, or class diagrams together with CRC-cards. The trade-off is that PVMs are often found to be easier to understand by domain experts, whereas class diagrams provide a more formal and richer notation, which is better for documentation of software aspects. A major challenge in this phase is to ensure that the configurator is further developed as the product assortment changes. If this is not done, the configurator will soon become of little use for the company.

4.5 Mass customization at ETO companies

ETO companies that apply configuration technology are often labelled mass customizers in configuration literature. In many cases, however, there seems to be a conflict between the popular definitions of 'mass customization' (i.e. involving mass production prices or efficiency) and the literature that claims that ETO companies become mass customizers. Most often, when ETO companies move toward mass customization, they do not necessarily become mass customizers in the sense that they are capable of producing products at prices close to what they would be if they were mass produced. Because of the increased standardization, however, they can be capable of delivering customized products at prices lower than traditional ETO companies. From a product price perspective, these companies can be placed somewhere between ETO and mass customization.

Haug et al. (2007) (found in appendix 1) argue that since being a mass customizer does not rule out that a company can also create mass produced products in parallel, a similar perspective can be applied to ETO companies, but instead with a focus on product design processes. From this perspective, it can be claimed that such ETO companies are mass customizers in certain parts of the product specification process, since the use of configurators can decrease the costs of some of the specification work associated with customized products to costs comparable to using standard product specifications. Such a pattern is seen in some ETO companies where some of the product solution space is predefined. This is often the choices in the early design phases, while detailed design decisions do not take place within a predefined solution space. Another perspective is to perceive customized product specifications (such as quotes) as a mass customized product, which can be produced at costs close to standard specifications through the use of configuration technology.

Chapter 5: Scientific approach

From my point of view, two important tasks in relation to the scientific aspect of a PhD project are to define the perception of science that forms the basis for the research carried out, and to position this perception of science in relation to existing lines of theory of science. Defining scientific assumptions before carrying out research guides the choice of research methods and serves to ensure consistency in the research carried out. The importance of positioning research according to different lines of scientific theory is related to communicative issues. For instance, when encountering interesting research results, knowing on which scientific assumptions these results have been based can give a quick indication of how to interpret such results and to evaluate whether they offer adequate validity on which to build.

In this chapter the scientific theoretical assumptions of the main part of the existing configuration research that this thesis builds on (i.e. CPM literature) are discussed. On this basis, the chapter proposes a shift from the existing scientific ideals to a perception that provides greater consistency between the research carried out and the underlying scientific theoretical assumptions. The majority of the content related to the science theories discussed in this chapter is based on secondary sources in order to ease the elaboration. The selected sources have been chosen from a broader range of literature to allow comparisons of interpretations to avoid one-sided or inexact generalizations, which is sometimes the case. A somewhat elaborate form has been chosen for this chapter in order to account for the complexity of the topic and the many and often differing views of science within individual lines of theory of science.

5.1 Basic concepts

Before engaging in the discussion of theory of science, it is necessary to understand a few essential terms, some of which are attributed slightly different meanings in different contexts. Some of the most central terms in a discussion of theory of science are: ontology, epistemology, hypothesis, theory, methodology, and research.

As a starting point, Brier (2006) provides an interesting model that places some of these terms in relation to each other, as shown in figure 17.

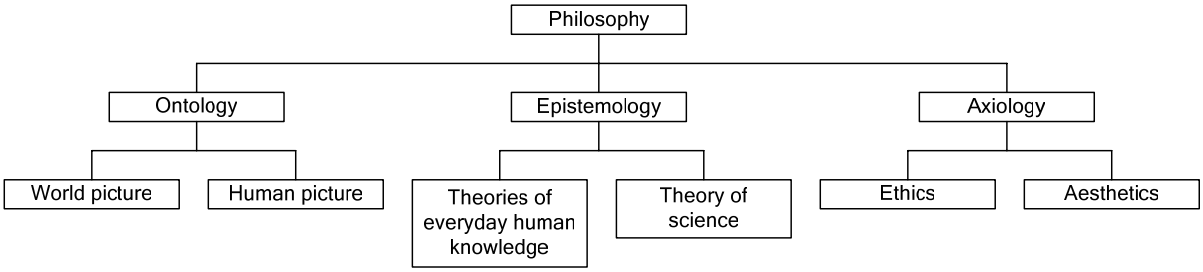


Figure 17: Philosophy (trans. from Brier, 2006)

There seems to be general agreement that a scientific hypothesis is a testable (at least in principal) working assumption that explains a phenomenon. However, the requirements for how hypothesises should be formulated, tested, and the results interpreted are different depending on the perspective on science. The concept of a theory ranges from something very abstract (close to philosophy) to more concrete sets of hypotheses of specific phenomena. A theory is an abstraction that describes certain aspects of a phenomenon, separated from other aspects that also can impact the phenomenon (Danermark et al., 2002). In brief, ontology is theory about what exists in the world and how it exists. Epistemology is theory about knowledge, i.e. about what we can know about the world and how we can know this. When asking if something actually exists, this belongs to the ontological field, while questions about objectivity and methods belong to the epistemological sphere (Buch-Hansen and Nielsen, 2007). Methodology deals with the knowledge about methods used to create scientific knowledge. A methodology includes a collection of methods and describes how and in which

situations these should be applied. A theory of science is more of an overall theory about science and research (Brier, 2006).

A traditional distinction can be made between 'basic research', 'applied research' and 'experimental development' (Brier, 2006). Basic research is original investigations with the main purpose of achieving new scientific knowledge, often systematic further development of our science systems through either theoretical work and/or new experiments. Basic research is not primarily targeted at practical applications, but is often aimed at descriptive, explorative and explanatory applications. Applied research is original investigations that aim at achieving new scientific knowledge and understanding, but primarily targeted at practical applications. Applied research can have diagnostic or problem-solving purpose, e.g. of technical, medical, communicative or organizational character. Experimental development is systematic work based on the application of already gained knowledge, for the creation of new or significantly improved products, materials, systems/machines or services.

5.2 Scientific approach of existing research

The scientific theoretical standpoint of the operation management section of Department of Manufacturing Engineering and Management at Technical University of Denmark, to which I belong, is based on critical rationalism (Svensson, 2003; Riis, 2003, Hansen, 2003, Malis, 2005). Fallibilism is also mentioned, although this is an implicit consequence of critical rationalism. Therefore, the natural starting point of this chapter is to investigate if the critical rationalistic perspective is reasonable in the context of conducting configuration research as it has been carried out at CPM till now, and in connection with my specific research questions.

5.2.1 Fallibilism, positivism and critical rationalism

'Fallibilism' was originally presented and named by the American philosopher, Charles Sanders Peirce (1839-1914), known as the earliest proponent of the philosophic school of pragmatism[9] (Lübcke at al., 2001). Fallibilism is the epistemological theory that postulates that scientific knowledge is always temporary, since it can later be changed. Fallibilism, therefore, implies that there are no self-evident truths, infallible observations, aprioristic principles etc. The idea of fallibilism was later adopted by Karl Popper (1902-1994) in critical rationalism, which advocates that development of scientific knowledge should proceed by using falsification rather than verification.

Critical rationalism is rooted in positivism. The term 'positivism' originates from the French philosopher, August Comte (1798-1857), and describes a particular view of what science is. The most thorough and modern form of traditional positivism is logical positivism, which emerged in the 1920s in Vienna with such persons as Schlick Rudolf Carnap (1891-1970), Hans Hahn (1879-1934) and Otto Neurath (1882-1945) as the most important figures (Pedersen and Toft, 2005). Like Comte, the logical positivists have the basic assumption that science is empirically founded, but they also include the use of logic and mathematics for organizing knowledge. However, being based on the idea of deriving scientific knowledge by induction, positivism faces a number of problems (Chalmers, 1999). One main problem is stating under which exact conditions a generalization is a sound inductive inference. This is because inductive arguments refer to prior knowledge, which also has to be derived by induction and based on new conditions, which also need to be justified, and so on. Another problem is inexact measurements. Although this is accepted by natural scientists, it raises the problem of how exact laws can be derived from inexact evidence. A third and well-known problem is 'the induction problem'. The induction problem refers to the fact that since general scientific laws go beyond a finite number of observations, these can never be proven. An often used example of this problem is that at one time all swans encountered were white, and the conclusion was therefore that 'all swans are white'. However, when the Australian continent was discovered, black swans were observed. Recognizing

[9] The criterion for 'truth' in pragmatism is that a theory or a model is truthful if it works in accordance with its purpose, i.e. the truth-criterion is connected with the concept of usefulness. In spite of sounding very practical at first glance, there is a great problem in this way of establishing 'truths'. For instance, a lie may be very useful in the short run. Peirce therefore ended up rejecting this interpretation of pragmatism, and instead founded what he termed pragmaticism, where the focus on truth changes to a focus on meaning (Brier, 2006).

that it may not be possible to establish absolute truths about reality, later positivists have resorted to the use of probabilities. Since, a finite set of observations divided by an infinite number of not yet conducted observations implies a probability going towards zero, more refined ways of dealing with probability have been created. However, none of these represent unproblematic solutions (Chalmers, 1999).[10]

Critical rationalism is an epistemological school founded by Popper. Critical rationalism emerged from logical positivism as a solution to many of the problems faced by logical positivism. The goal of critical rationalism is to define how empirical science shall be carried out in order to achieve scientific growth. Critical rationalism overcomes 'the problem of induction' by the application of the principle of falsification instead of verification. This means that only negative test results tell us something absolute about a theory, and that theories will always be speculative and can never be considered absolute truths. In this connection, Popper advocates that a hypothesis consist of bold, highly testable statements (Chalmers, 1999).

The line of thought in critical rationalism applies a strict definition of the concept of a 'theory.' Popper defines an acceptable 'empirical theoretical system' as fulfilling four conditions (Koch, 2005, based on Popper, 1934/1980):

- It consists of a set of formalized statements (if they are not formalized, they cannot be used for deductions)
- The set should be mutually consistent (i.e. no contradicting statements)
- Some of the statements must be synthetic (the truth of the predicate may not be a consequence of the nature of the subject, e.g. in an analytical statement such as 'all bachelors are not married')
- The set of statements should at least in principle be falsifiable (at least one potential event exists with which the statement is consistent)

This view of science contrasts with many other fields of research, and Popper has for instance criticized the psychological theories of Sigmund Freud (1856-1939) and Alfred Adler (1870-1937) for being so imprecise that these are impossible to falsify (Chalmers, 1999).

In spite of seemingly providing a solution to some of the problems of logical positivism, the principle of falsification has some significant problems. A principal problem is the case where some experiment or observation is in conflict with a hypothesis. The reason why a hypothesis is falsified may be that the evidence or the preconditions are faulty, rather than the hypothesis itself. Therefore, a hypothesis is not necessarily falsified because some observation or experiment conflicts with it. The history of natural science holds significant examples of this, e.g. Newton's gravitational theory and Bohr's theory of the atom were initially falsified because of unknown factors, which were discovered later (Chalmers, 1999). Because of the problems with the degree of definiteness with which theories can be falsified, Popper also admitted that it is often necessary to retain theories, although apparent falsifications exist. Obviously, this confession raises the question of what then is left of this kind of falsificationism. Not only has Popper tried to adapt critical rationalism to deal with its problems; also Imre Lakatos (1922-1974) has further developed what he calls the 'naive critical rationalism' of Popper to a 'sophisticated critical rationalism' in an attempt to answer some of this criticism.[11]

Positivist notions have also been challenged by Thomas Kuhn (1922-1996) in his book, "The structure of scientific revolutions" from 1962, which has been one of the most influential in discussions about theory of science. Kuhn presents the central idea that science is developed within incommensurable paradigms, which implies that fruitful scientific discussions are not possible across paradigms (Koch, 2005). An important point made by Kuhn is that positivist notions (including critical rationalism) do not comply with historical evidence, but that science evolves through revolutions that cause existing scientific theories to be replaced by new ones. When a new scientific view becomes well-structured and relatively unchallenged, it becomes a 'paradigm'. The persons operating within this paradigm practice what Kuhn terms 'normal science'.

[10] E.g. 'Bayesian approaches', where e.g. some are criticized for implying subjective probabilities.

[11] E.g. by redefining when and how a negative result should be considered unambiguous, correct and solid enough to falsify an established theory, if a better theory does not emerge at the same time (Brier, 2006).

Another interesting point has been made in 'Foundations of social sciences' from 1944 by Otto Neurath, who points to the paradox that epistemology faces, namely the problem of circularity. Any epistemology presupposes knowledge of the conditions in which knowledge takes place, but since science cannot be used in order to ground the legitimacy of science, there are no secure foundations from which we can begin any consideration of our knowledge of knowledge. From this perspective, what we have are rather competing philosophical assumptions (Johnson and Cassel, 2001).

The unfortunate conclusion, when looking at the results of physics and the application of scientific methods, is that most of the episodes in history that are seen as major advancements, e.g. the innovations of Galileo, Newton, Darwin and Einstein, do not correspond to the standard philosophical accounts of science (Chalmers, 1999). A possible implication of this recognition would be to give up the idea that science is a rational activity that operates according to some special method. This kind of reaction is found in philosopher Paul Feyerabend's (1924-1994) book, "Against method: Outline of an Anarchistic Theory of Knowledge", from 1975. In brief, the position regarding science Feyerabend presents is provocatively formulated as 'anything goes', and makes the point that the choice of scientific theories is determined by subjective values and desires. The position of Feyerabend is a sign of extreme relativism, which dissolves the concept of scientific objectivity. The scepticism of Feyerabend about the attempts to rationalize science is shared by much recent writing from social science or so-called postmodernist[12] perspectives (Chalmers, 1999).

5.2.2 Discussion of critical rationalism in configuration research

Belonging to a research tradition that subscribes to critical rationalism, a basic question of this PhD thesis must be whether the critical rationalistic view of science corresponds with my perception of science. While the research tasks of this PhD thesis do focus on techniques and methods for solving isolated problems, much of the research produced at CPM focuses on developing procedures to be used for changing specification processes at companies. Therefore, it may seem that I in principle could disregard the scientific approach of the research in the CPM-procedure(s) and focus on defining a scientific approach that supports my particular research areas. However, in order to base my work on results of research conducted at CPM, it would not be reasonable were I to subscribe to a theory of science that implies a categorization of the research approaches applied at CPM as unscientific. This means that my scientific approach should be able to include both existing CPM research and my own specific research. For this reason, the following discussion of the scientific approach of CPM research should not be considered as criticism; on the contrary, it is rather a justification of the scientific quality of existing CPM research.

Critical rationalism and the CPM-procedure

According to critical rationalism, valid hypothesises must be falsifiable by logically possible observations. Statements such as 'the use of the CPM-procedure can lead to significant benefits' or 'the use of the CPM-procedure most often leads to significant benefits' are therefore not valid, since no observation statement can falsify such a hypothesis. On the other hand, claiming that 'the use of the CPM-procedure always leads to successful projects' (e.g. in the sense that a configurator is in operation within 5 years) can and in fact has been falsified. The cases where the use of the CPM-procedure did not result in a configurator being implemented in an organization can be dismissed by claiming that, in these cases, it was not due to the fact that the CPM-procedure did not work but because of other factors. Therefore, to avoid falsifications of a hypothesis such as 'the use of the CPM-procedure leads to successful projects', it is necessary to define criteria for the hypothesis that ensure that the hypothesis does only apply to the situations where it holds. However, identifying such criteria for measuring 'when it works' and 'how it works' in configuration projects seems like an impossible mission. The problem of testing the CPM-procedure from a critical rationalistic perspective is that the applicability of the procedure depends on human factors. This implies that the background conditions

[12] Post-modernism: "Since the mid-1970s an influential cultural and philosophical direction that is in radical opposition to positivism and modernistic ideals about absolute truth, reason and universal liberation. The preferred method of postmodernists is deconstruction, and they perceive the social reality as being fragmented, without depth, contingent and history-less." (trans. from Buch-Hansen and Nielsen, 2007).

are not just physical or physiological facts, but at times irrational human behaviour. Humans are not predictable in the same manner as other aspects of nature (e.g. the law of gravity), and for this reason, examples can always be found to counter generalized statements about human behaviour. Therefore, theories about the applicability of the CPM-procedure in organizations cannot be described by a set of rules in a way that is not easy to falsify, and for this reason such 'theories' are not really theories in the traditional sense.

To avoid drifting too far away from the existing scientific ideals of CPM, a temptation could be to use a quantitative approach with a vague form of verification that accepts that exceptions can occur. For example, it could be stated that 'the application of the CPM-procedure can lead to configurators being implemented in an organization with positive results', together with a definition of positive results; i.e. not a valid hypothesis in the traditional sense. However, it is also possible to find many other cases where a configurator has been created by using other procedures, for which reason the theory may just as well be broadened to 'many procedures for developing configurators can lead to configurators being implemented, including the CPM-procedure'. Thus, we have not really learned anything significant from this kind of study. Making such studies interesting by comparing success-rates of projects where the CPM-procedure has been applied with projects where other procedures have been applied could seem like an answer. However, as mentioned, many other factors than the application of the CPM-procedure could affect the result. Thus, such studies may even show that other procedures are far better without this necessarily being the case.

Conclusion of discussion

If to test the applicability of the CPM-procedure means to eliminate or fully control social factors, this is an impossible mission, since the procedure is applied in social contexts. Also, it seems to be almost impossible to define criteria for hypotheses that could say anything interesting about the CPM-procedure based only on quantitative data while not being able to find cases that counter such conclusions. Therefore, it seems to be extremely difficult, to say the least, to say anything relevant about the CPM-procedure from a critical rationalistic point of view. However, this is not to say that I do not believe that the CPM-procedure is a good approach; on the contrary, it is my belief that it may be the procedure that provides the solid foundation for the development, implantation and maintenance of product configurators based on standard software. My point is merely that critical rationalism cannot be used to support this belief.

Given this paradox of subscribing to a theory of science that does not allow knowledge to be produced about the object in focus, the question is how existing research deals with this. The main way of investigating the CPM-procedure, together with related methods and techniques, has been to conduct case studies, often in the form of action research. Here, the first conflict emerges, because action research must be said to be one of the least objective methods and is therefore in striking contrast to critical rationalism and other positivist notions. Next, experience with the use of the CPM-procedure has resulted in many changes from the time it was introduced till the present (Hvam, 1994; Hvam et al., 2007a). Through its evolution, the CPM-procedure has been continuously applied in industry, and experiences with its use have been gained. The changes in the CPM-procedure can be explained by the discoveries of what have been perceived as new and better ways of doing things, which have been integrated into the procedure. However, this development of the procedure and the growing knowledge about product configuration in no way resembles the instructions of Popper's critical rationalism concerning how to produce scientific knowledge. Therefore, subscribing to critical rationalism as a theory of science can be, in the context of analysing the CPM-procedure, not much more than an ideology, while what is done in practice is something significantly different, even contradictory. Subscribing to the scientific ideal of critical rationalism is in many ways the same as saying that the knowledge produced about the CPM-procedure is not scientific knowledge. In such a case, it would probably be better to not deal with theory of science at all, and only focus on research techniques when carrying out research. However, from my point of view, such an approach would not be satisfactory.

In the cases studied where the CPM-procedure has been applied, the effect of using the procedure has been dependent on the context in which it has been applied, i.e. such factors as management support of the project, qualifications of project participants, product assortment complexity, resources available etc. These determinant factors include aspects that quantitative measures alone cannot

capture. Something similar applies for the methods, techniques and tools applied in configuration projects; also their success often depends on human factors. To give a very concrete example, the use of class diagrams may produce better results for one particular social group, while PVMs may produce better results for another. This may be explained by the assumption that educational background, work experience, psychological factors etc. of the different groups favour different notations. Such aspects are hardly measurable in an unambiguous fashion. On the other hand, the alternative of only relying on qualitative factors can also be problematic. For instance, if a person claims that he finds one graphical notation better than another (e.g. faster to learn and resulting in fewer errors), the problem is that facts cannot be separated from values. Quantitative investigations may show or indicate that another notation is faster to learn and results in fewer errors for the specific person, while he still claims to prefer the first. My argument is, therefore, that both quantitative and qualitative factors can be highly relevant in this line of research.

This leads to my conclusion that the best way of producing scientific knowledge in relation to the CPM-procedure and the techniques applied is to reject positivist notions, and allow interpretation of the experiences from using the procedure. Although such an approach corresponds well with how research is actually carried out when studying the use of the CPM-procedure, this represents a much different scientific perception than what has been described so far.

5.3 Alternative scientific approaches

The arguments presented for the importance of context in studies where social factors have an impact on the research objects are not accepted by all researchers, however. Unfortunately, it frequently occurs that 'empirical objectivism' is applied to fields where unpredictable social aspects have a significant impact, which, therefore, results in arriving at highly questionable or at least easily falsifiable conclusions. This tendency can be explained in part by the appeal of the natural science ideal, which, compared to social scientific theories, builds on simple logic and in some ways have had great success in producing knowledge and technological progress. In such a perspective, some social and human sciences (also natural sciences) apply the argument that the undoubted success of physics during the last 300 years, for example, can be attributed to the application of 'scientific method'. Therefore, to achieve the success of natural science, social and human sciences should apply the scientific method of natural science (Chalmers, 1999). However, after more than 200 years with natural-science-modelled social sciences, such approaches can still not be categorized as 'normal science' (in Kuhnian terminology), since social science has still not moved in the desired direction, namely towards predictive theory (Flyvbjerg, 2001; Danermark et al., 2002). This raises the question of whether it is reasonable to apply natural scientific method elsewhere. From my point of view, it is more constructive to accept the great differences between the two fields of study. Therefore, the next step in this thesis is to investigate alternatives to the application of critical rationalism in my line of configuration research.

For an overview of different lines of scientific theories, Fuglsang and Olsen (2005), not without reservations, divide scientific theories into 'demarcationisms', 'scientific realisms' and 'complex idealisms'. Demarcationisms are defined as scientific approaches that have as their main purpose to separate scientific from non-scientific approaches and to derive empiric regularities. Logical positivism and critical rationalism are included in this category. Scientific realisms are approaches that in contrast to demarcationisms assume that interpretations are necessary in order to uncover objective structures or connections lying underneath the observable reality. Included in this category are: systems theory,[13] critical realism, Marxism, critical theory, and sociologic field analysis.[14] Complex idealisms are described as approaches that strive to revoke the contrast between subject and object, and assume that the society we deal with consists of webs of thoughts and materiality.[15] In this category are placed: phenomenology, hermeneutics, social constructivism, discourse theory, actor-

[13] Not to be confused with general systems theory.

[14] Fuglsang and Olsen recognize that early phenomenology and methodical hermeneutics could have been placed here.

[15] In contrast to the naive realists, for whom experience builds on sense data, and naive idealists, for whom experience only builds on thoughts.

network theory, and organization.[16] In addition, Fuglsang and Olsen mention feminist theory and action research, which do not fit directly into the mentioned categories.

Since I concluded early in the course of my PhD project that my scientific ideal could not be found in what Fuglsang and Olsen call demarcationisms, I have considered alternative approaches. At the beginning of the work on my PhD, my focus was on systems theory and (social) constructivist approaches, and later I encountered critical realism. For this reason, these approaches are described and discussed in the following sections. On the other hand, hermeneutics[17] and phenomenology[18] are not explicitly dealt with, even though they might be the most important meta-theoretical starting points of qualitative methodology (Danermark, 2002). This is primarily in order to limit the extensiveness of this chapter. Also, I later deal to some extent with such meta-theoretical considerations in connection with the definition of my perspective on theory of science.

5.3.1 Systems theory

Systems theory is an obvious place to start in the search for an alternative to critical rationalism, since such theory forms an important basis for the way that companies and products are described by my recent predecessors, who have dealt with product configuration at Department of Manufacturing Engineering and Management at Technical University of Denmark (Svensson, 2003; Riis, 2003, Hansen, 2003, Malis, 2005).

General systems theory

The term 'system' typically refers to something that is composed of elements, and where the whole is more than the sum of its parts. In the evolution of science, systems theory in a well-defined sense emerged relatively late, beginning in the 1930s, and marked a scientific renewal from classical analytical research (Kneer and Nassehi, 2004). The changed perception of nature was especially based on the biological critique of classical physics. The Newtonian way of describing reality implies that reality can be described mathematically and deductively; however, this kind of explanation is problematic in biology, since the central object of biology, life, cannot be described this way. In biology, the focus is not on single phenomena, but on an established network of single phenomena; i.e. when looking at living organisms, the relations between its elements are essential. Partly responsible for this change of paradigms is the zoo-biologist Ludwig von Bertalanffy (1901-1972), who can be considered the father of the interdisciplinary general systems theory (see e.g. Bertalanffy, 1972).

Bertalanffy did not consider his work to be a general super-theory that includes general regularities for describing the phenomena of the world with systems theoretical concepts. But he believed that this theory could both extend the Newtonian world picture and be applied in areas that are not directly explained by physical and chemical laws, but have regularities that can be described by appropriately chosen model concepts (Kneer and Nassehi, 2004). This is in some ways confirmed in academia of today, where system theoretical perceptions are found in very different scientific disciplines, such as economics, medicine, pedagogics etc., but to less extent in the exact sciences. In the judgement of Kneer and Nassehi (2004), the content of the interdisciplinary applications of systems theory is not the common factor; rather, it is the application of system theoretical structures.

Two central distinctions by bnm are those between organized complexity vs. unorganized complexity and open systems vs. closed systems (Kneer and Nassehi, 2004). Unorganized complexity means that single phenomenons are coupled linearly ('A' implies 'B' implies 'C' etc.), while organized complexity means that there is interplay between different phenomena. For this reason, a phenomenon cannot per definition be perceived as being caused by another phenomenon (linear coupling), but just as well as a consequence of interplay between several phenomena.

[16] Fuglsang and Olsen recognize that the systems theory of Niklas Luhmann could have been placed here.

[17] Hermeneutics may roughly be described as "'having the fundamental traits of focusing on the interpretation of meanings" (Danermark et al., 2002). A basic distinction should be made between methodical, philosophical (Hans-Georg Gadamer), and critical hermeneutic (Jürgen Habermas and Paul Ricoeur).

[18] Phenomenology provides a correcting position in relation to the idea of full objectivity by arguing that human cognition builds on experience (Fuglsang, 2005). Modern phenomenology is considered to be founded by Edmund Husserl around year 1900, and later developed further by his student, Martin Heidegger (Brier, 2006).

A closed system is defined as being internally stabile when achieving its equilibrium state, and it does not exchange elements with its environment. An open system does not necessarily achieve such an equilibrium state (only temporarily) and has inputs and outputs, whereby the elements of the system consist of changes. The ability to change interrelations between elements makes it possible for an open system to survive, in spite of losing and gaining elements. Bertalanffy's idea means that there is no linear causality between the system and its environment, i.e. a particular input does not allow only one reaction. Since open systems internally reorganize their elements when the environment changes, open systems are generally classified as self-organizing systems (Kneer and Nassehi, 2004). The concept of systems themselves controlling their inner processes according to their own inner logic is captured in the term 'autopoiesis'[19] that originated in the 1970s from biologists Humberto R. Maturana and Francisco J. Varela.

Sociological systems theory

Fuglsang (2005) defines three directions within systems theory: a functionalistic perspective (Talcott Parsons), a pluralistic perspective (Robert K. Merton; Jeffrey C. Alexander) and a constructivist perspective (Niklas Luhmann). He further argues that within these types of (sociological) systems theories, the debate in Denmark is very limited, except for Luhmann, who is in focus in certain parts of academia. The extensive criticism that the systems theory of Parsons has been subjected to and the vagueness of the pluralist approach (both described by Fuglsang, 2005) may explain why it is Luhmann who has received the most attention in Danish academia in recent years. Also during the course of my PhD project, the theory of Luhmann is the only one of the mentioned lines of social systems theories that I have used significant effort to investigate.

My impression of the theory of Luhmann is that it is rather extensive and complex, not least because it includes many redefinitions of concepts normally used otherwise. This impression is not uncommon (Fuglsang, 2005; Mortensen, 2004; Thyssen, 2003). To give some examples of redefined concepts: In Luhmann's theory, it is not humans that communicate but the communication itself, and humans are not part of social systems, but the environment of social systems (Mortensen, 2004). To be able to understand Luhmann, therefore, requires some familiarity with many such redefined concepts that normally have other meanings. Furthermore, Luhmann is a social scientist in the theoretical tradition where empirical investigations are not in focus, but rather the development of concepts for describing society (Thyssen, 2003). Because of this focus, I concluded that although I find the ideas of Luhmann interesting and may in the future apply some of its elements, I do not perceive the theory of Luhmann as a complete alternative to the existing scientific theoretical perceptions of my department. For this reason, the theory of Luhmann is not discussed further in this context. For a gentle (if that is possible) introduction to his theory, I recommend Luhmann (2003) and Kneer and Nassehi (2004).

Discussion

As mentioned, the general systems theory has not had the big impact on the exact sciences that Bertalanffy had hoped for. But general systems theory has had a significant impact on many fields with very different focus and philosophical assumptions. Systems theory in itself is not a complete scientific approach but more of a basic language for organizing information about the world. It does not tell us what science is, but it can and has often been combined with various scientific assumptions. Therefore, Fuglsang (2005) finds it hard to say anything coherent about the methodology and commonly applied research techniques for a systems theoretical approach, since it is not primarily an empirical theory. This is also reflected in most books about theory of science, where systems theory is rarely included as a scientific theoretical direction of its own. In line with this discussion, I apply systems theoretical descriptions of phenomena, but I also recognize that this in itself is not an adequate scientific basis, and move on to other approaches.

[19] Autopoiesis refers to self-creating and self-organizing autonomous systems, and the term is commonly used e.g. in contexts describing biological and human organizations.

5.3.2 Social constructivism

An obvious reaction to my rejection of the positivist ideas of science could be to take a position at the opposite end of the scientific spectrum, namely social constructivism. A scientific approach that combines critical rationalism and social constructivism has been suggested in an earlier PhD project from CPM (Hansen, 2003), based on the recognition that basing research about procedures similar to the CPM-procedure on critical rationalism is hardly possible. But although I recognize that both critical rationalist and social constructivist approaches can to some degree be useful for studies in separate areas, I do not believe that they can be combined in a common scientific framework. The contradictions of these approaches are simply too fundamental. Therefore, subscribing to constructivism would have to be an alternative to critical rationalism.

Basic perceptions of social constructivism

Within traditional epistemology, with its belief in pure objective cognition, social structures are only considered for the purpose of explaining errors and shortcomings. On the other hand, the objective of social constructivism is to ask how knowledge is created instead of if it is true or valid (Wenneberg, 2002). On this question, social constructivism argues that our cognition is not independent of the social contexts that we are part of. Social constructivism can be characterized by the belief that our reality is shaped in a significant way by our cognition of it, as well as the assumption that phenomena in society are shaped by historical and social processes. From the perspective of science theory, social constructivism can be placed in opposition to realism, which believes that the reality is something objective that exists independently of our cognition of it.

The social constructivist idea can be traced back to Immanuel Kant (1724-1804) in his confrontation with radical empiricism (e.g. David Hume (1711-1776)), when he claimed that language comes before reality, for which reason the structure of language determines what facts are (Rasborg, 2005 from Hartnack, 1979). Later, Friedrich Nietzsche (1844-1900) presented an even more radical cognitive critique that attacks the idea that cognition can criticize its own basis, i.e. all description of reality is an interpretation and no objective truths exists (Rasborg, 2005 from Schmidt, 1984). In more recent sociology, social constructivism also plays a significant role in the work of important social scientists such as Ervin Goffman (1922-1982), Peter L. Berger (b. 1929), Thomas Luckmann (b. 1927) and Pierre Bourdieu (1930-2002) (Rasborg, 2005).

Social constructivism is not an ambiguous concept, since many social constructivist positions exist. To start with, it is possible to identify some perceptions which, with different emphasis, are part of theories that can be considered to represent social constructivism (Rasborg, 2005):

1) Anti-essentialism (no predefined nature decides the shaping of the individual and society)
2) Anti-realism (our knowledge is not a reflection of reality, but an interpretation)
3) The always historical and cultural character of knowledge (knowledge is always decisively affected by social and cultural context)
4) The primacy of language compared to thinking (the content of the language is decisive for what we are capable of thinking)
5) Language as action (saying something is the same as doing something)
6) Focus on interaction and social praxis (social processes are constituted by social praxis)
7) Focus on processes (in analysis of social phenomena, the focus should be on processes instead of structure)

Collin (2003) argues for two basic distinctions that are important for describing various lines of social constructivism, namely epistemological constructivism vs. ontological constructivism, and physical reality vs. reality of humans and society. The epistemological constructivist perspective implies that the contents of scientific theories are solely, or for the most part, determined by social factors that surround the research process, while the ontological perspective implies that reality is not independent but constituted by the cognition that it is subjected to, i.e. an even more radical point of view. Both perspectives can be attached to the physical reality and the human/social reality. While both the epistemological and ontological social constructivist perspectives may seem rather radical, Wenneberg (2002) defines two more moderate types of social constructivism, a critical perspective and

sociological theory perspective. In brief, the critical perspective can be defined as natural scepticism toward taking the natural for granted (e.g. certain natural human actions in a particular society), while the sociological theory perspective applies a critical perspective to social institutions (e.g. Berger and Luckmann[20]). To illustrate the connection between social constructivism and realism, Wenneberg defines three different types of epistemological perspectives:

1) Naive realism/Anti-constructivism
2) Nuanced realism/Trivial constructivism
3) Anti-realism/Radical constructivism

The first perspective represents the idea that reality affects science in an unquestionable and direct way; the second perspective represents the idea that science is constructed through both reality and social factors; and the third perspective represents the idea that science is only a result of social factors.

Most often, it is the most radical variants of social constructivism that are criticized, which may be a bit unfair to more moderate social constructivist perspectives. The basic problem that is attributed to social constructivism is relativism. In brief, the problem of relativism is that if knowledge is determined by other factors than the reality it deals with (i.e. psychological and social factors), an instance which can serve as the judge of the question of truth is absent. If the judgement of truthfulness is made relative, then any knowledge can be as good as any other knowledge, and we can know nothing (Wenneberg, 2002). This also implies a self-contradiction, since if it is claimed that no general truths exist, relativists cannot make any such claims either. This implies that in principle what relativist researchers are doing cannot be qualified as scientific or even meaningful.

From my perspective, a particularly interesting line of (social) constructivism is the one dealing with sociology of technology. Of important contributions can be mentioned the SCOT (Social Construction Of Technology) by Trevor Pinch and Wiebe Bijker (e.g. Pinch and Bijker, 1984) and ANT (Actor Network Theory), where the most important persons are Michel Callon, Bruno Latour and John Law (e.g. Callon, 1987; Latour, 1987; and Law, 1991). ANT is normally defined as a 'constructivist' and not a 'social constructivist' approach, since in this line of theory humans are not the only defining actors, but so are physical and conceptual elements (sometimes named 'actants').

Discussion

As mentioned, my problem with logical positivism and critical rationalism is the idea that science can only be conducted according to one method, which may fit very well within natural science but cannot produce much knowledge when investigating areas in which social elements play a role. In such contexts, it seems that one is forced to disregard highly relevant observations or understandings for the sake of the method, which I find very unhealthy. At the other end of the science theory spectrum, social constructivism provides a solution by recognizing the importance of social factors. From a scientific point of view, I accept the many of the ideas in a moderate form of social constructivism. But, on the other hand, I find it important to emphasize my focus of investigating a reality that I perceive as existing independently of our cognition of it. In this respect, I therefore share the realist perspective with positivist notions. Since I focus more on explaining the phenomena I observe, rather than focusing on how social factors affect knowledge construction, I find that labelling myself as a social constructivist would give a somewhat misleading impression, which I would like to avoid. Furthermore, it is important for me, while recognizing the importance of social context, not to get caught in the trap of not being able to generalize my findings. The problem of not being able to make generalizing claims is according to Flyvbjerg (2001) a significant problem of much social scientific research.

To sum up, I basically recognize some elements of both the two extremes: positivism (the realist perspective) and social constructivism (the influence of social factors) and disagree with others. Therefore, I would be very reluctant to subscribe to any of these views or to try a combination, since

[20] Peter L. Berger and Thomas Luckmann have had a significant impact on the modern sociology of knowledge (e.g. organization theory), where an important work is "The social construction of reality, a treatise in the sociology of knowledge" (1987).

as mentioned, in principle, they rule each other out. Based on these recognitions, the line of theory of science named 'critical realism' seems to be an obvious perspective to investigate.

5.4 Chosen scientific approach

5.4.1 Critical realism

The emergence of critical rationalism (as a theory of science) can be seen as starting in 1975, when philosopher of science, Roy Bhaskar, developed the concept of 'transcendental realism' in the book, "A Realist Theory of Science" (Jespersen, 2005). Bhaskar later developed the perspective of 'critical naturalism', which with accept of Bhaskar was merged with 'transcendental realism' into 'critical realism'. Critical realism has its inspiration from the scientific theoretical discussions of natural sciences. But while having originally been developed as an alternative to positivism, critical realism later also began to illuminate sociological and economical perspectives. Some of the most important that can be mentioned are Margaret Archer's "Realist Social Theory - The morphogenetic approach" (1995), and Tony Lawson's "Economics & Reality" (1997). Critical realism can be seen as a holistic alternative to both positivist and postmodernist approaches.

Critical realism is therefore a rather new scientific approach, which has not yet found its final shape. Texts in Danish that deal in detail with critical realism have only emerged in the most recent years, but with rapid growth (Buch-Hansen and Nielsen, 2007). Since the beginning of the 1990s, Bhaskar has been further developing critical realism in new surprising directions, which has made the picture of critical realism even more complex. Critical realism is therefore not a homogeneous movement, but it includes some differences in perspectives and developments (Danermark et al., 2002; Buch-Hansen and Nielsen, 2007). To present all such discussions would be too extensive and not particularly relevant in this context; therefore, in the following, I draw out the main ideas and lines of thoughts that I find particularly important for my research. But before going into detail about critical realism, a natural starting point is the claims of Danermark et al. (2002) about the perceptions that critical realism helps us to argue for:

1) "Science should have generalizing claims."
2) "Explanation of social phenomena by revealing the causal mechanisms which produce them is the fundamental task of research."
3) "In this explanatory endeavour abduction and retroduction are two very important tools"
4) "The role of theory is decisive for research."
5) "Research involves a wide range of methodological tools, and we have to use many of these tools in a concrete research project. In other words, there is often a need to mix methods."
6) "There is a need to overrule the categorizing of methods in quantitative and qualitative terms."
7) "The nature of society as an open system makes it impossible to make predictions as can be done in natural science. But, based on an analysis of causal mechanisms, it is possible to conduct a well-informed discussion about the potential consequences of mechanisms working in different settings."

Ontological and epistemological basis

Realist science theories, i.e. both positivism and critical realism, share the common belief that reality has an objective existence. The point on which such theories disagree is on the nature of reality. As opposed to positivist notions, critical realism does not believe that we have full access to reality. This means that reality is not transparent but has mechanisms that we cannot observe and only experience indirectly by their ability to cause phenomena (Danermark, 2002). Therefore, critical realism is fundamentally different from logical positivism and other positions based on positivistic assumptions,[21] and in his early works, Bhaskar attacks the positivistic kind of realism, which he calls

[21] In spite of the attacks on the logical positivism of Popper, typically critical realists consider critical rationalism to be a special kind of positivism (Buch-Hansen and Nielsen, 2007), a perception shared by much social science.

'empirical realism', for holding the perspective that the reality is only what is observable by human senses (Buch-Hansen and Nielsen, 2007).

The fundamental characteristics of critical realism can be described as: "Critical realism claims to be able to combine and reconcile ontological realism, epistemological relativism and judgemental rationality" (Danermark et al, 2002 cf. Archer et al., 1998). Besides acknowledging an independent reality, the statement asserts that all knowledge is produced in social contexts based on existing knowledge. This epistemological relativism could sound like a fall straight into the pitfall of methodological relativism. However, the solution to avoiding this relativism is arguing for our capacity for rational judgement, where the objective standard against which to measure scientific knowledge is the independent reality (Buch-Hansen and Nielsen, 2007). It is therefore central to critical realism that we are able to judge the strengths and the weaknesses of research methods. As formulated by Danermark et al.:

> "Such judging is best done from well-grounded metatheoretical assumptions. This often leads us to require a combination of several different methods. This mode of combining must, however, be based on ontological considerations. It cannot, as has been maintained by certain method pragmatists, be based on practical considerations and on empirical conditions." (Danermark et al., 2002)

Therefore, critical realists, like post-modernists, are epistemological relativists, but the significant difference is that while postmodernists also make the ability to judge relative, critical realists do not. Rational judgement means that although knowledge is relative and fallible, knowledge is by no means equally fallible. Leading theories can therefore only be seen as the 'best' theories about reality currently available, and they can always be replaced by better theories. If more theories about some phenomena exist, the theories may differ but not the reality they describe. Facts are therefore theory-dependant but not theory-determined (Danermark et al., 2002). This also targets Kuhn's claim of incommensurability between scientific paradigms, which is something that critical realism is sceptical about, because of not accepting judgemental relativism (Danermark et al., 2002).

The levels of reality

Within natural science, a need for a more spacious research programme emerged at the end of the last century because of Einstein's theory of relativity and quantum mechanics, which both delimited the domain of classical physics, since classical physics was not able to explain the speed of light or the apparently random behaviour of electrons. This urged the extension of the ideal model through new research programmes; not to make classical physics superfluous but to describe hitherto unrecognized structures of the analysed reality. According to Bhaskar, such events are good examples that show that beneath the recognized reality, there is a sub-world of transcendental phenomena (Jespersen, 2005).

In the book, 'A realist theory of science', Bhaskar provides a distinction between three ontological domains: the empirical, the actual and the real. The empirical domain comprises our direct and indirect experiences. In the actual domain, events happen whether observed or not; and what is observed is not the same as what actually happens in the world. In the real domain, generative mechanisms and structures that can produce events in the world are located (Danermark et al., 2002). This ontological perspective is shown in figure 18. The definition of the 'real domain' reflects a clear difference between critical realism and positivist notions.

1. Empirical domain: 2. Actual domain:	Experience Phenomena	Imprecise measurements Conditional predictions
3. The real domain:	Mechanisms	Open theories/hypothesises

Figure 18: Critical realist ontology (right column from Jespersen, 2005, based on Lawson, 2003)

From a critical realist perspective, the goal of science can be seen as trying to transform scientific theories of the independent reality into deeper knowledge of reality. Science must establish connections between the three domains in order to observe and identify the effects of the mechanisms

of the real domain. The larger a domain of study is on the two upper levels, the more empirically based knowledge can be acquired, but always with uncertainties related to data and the contents of the real domain (Jespersen, 2005).

One of Bhaskar's central points is the clear distinction between two dimensions of science, the 'transitive dimension' and the 'intransitive dimension'. The transitive dimension includes our knowledge about the world that is available at a particular point of time; i.e., the creation of knowledge is a historical and human activity that occurs in social contexts. The intransitive dimension includes the objects of the reality that science aims to create knowledge about. The intransitive dimension, therefore, exists independently of human knowledge about it. Critical realists see the intransitive dimension as more fundamental than the transitive; i.e., the reality is more fundamental than our knowledge about it (Buch-Hansen and Nielsen, 2007).

Mechanisms

Critical realists argue that the structures and mechanisms of the reality exist in some kind of order, where the structures and mechanisms of higher levels presuppose those of lower levels. This division of reality into levels is seen as unlimited, and can therefore not be defined in an absolute manner. But to illustrate the principle Buch-Hansen and Nielsen (2007) define the levels: social reality, which presupposes the biological level, which presupposes the chemical level, which presupposes the physical level. Events at higher levels may be effects of phenomena at lower levels.

Bhaskar deals with a reality that includes objects with different causal potentials. That an object possesses causal potentials does not mean that these are automatically activated to create an event, but this depends on the conditions of a particular context. What in critical realism is named 'mechanisms' decides whether causal potentials are activated. For instance, a car has the potential of driving, a dog has the potential of barking, and water can put out fire, but this does not mean they do these things, since something (at some level of reality) has to activate them (Buch-Hansen and Nielsen, 2007). When mechanisms generate an event, what is experienced becomes an empirical fact. But although an object has a causal power to do something and a relevant mechanism 'attempts' to trigger it, the right circumstances have to be present in order to create an effect. For instance, a match will not light if there is no oxygen available. Also, mechanisms can neutralize the effect of other mechanisms. For instance, when a bird flies, gravity is triggered, but the effect fails to appear because of other mechanisms (Danermark, et al., 2002). Finally, specific behavioural tendencies of an object can also be reinforced by behaviour of other objects. Objects can therefore only be attributed a tendency to act in a particular way.

All in all, the focus on mechanisms implies a switch to trying to understand what causes events rather than focusing on the event itself, i.e. an understanding of causality that differs dramatically from the idea that causality concerns empirical regularities (Buch-Hansen and Nielsen, 2007).

Social phenomena

In critical realism, the ontological perspective is also applied to the social world, implying that critical realists believe that the structures of the social world may at any time confront humans as an objective phenomenon (Buch-Hansen and Nielsen, 2007). But an important difference between natural science and social science is that while objects of both natural and social science are socially defined, natural objects are naturally produced, while social objects are socially produced (Danermark et al., 2002). This means that in natural sciences, it is sometimes possible to create an artificial closure of a system in experiments, while social systems are always open and cannot artificially be closed in the same way. However, the mutual interplay between the social sciences and the surrounding society does not mean that the boundary between the intransitive and transitive dimension should be dissolved; instead the social objects in focus always exist intransitively at the time they are studied (Buch-Hansen and Nielsen, 2007). While critical realists have the perception that social structures can be equivalent to the structures of nature, they acknowledge at the same time that social phenomena are something special. For this reason, the statements that are part of explanations of social phenomena should not resemble explanations from the natural sciences; i.e., there should be a difference in the way such statements are presented and tested (Buch-Hansen and Nielsen, 2007). Context-dependant knowledge and the openness of domains where social factors are significant give such sciences basic limitations that

cause generalized knowledge to be rather uncertain. For this reason, the scientific challenge may be to produce "good reasons to believe" (Jespersen, 2005, based on Lawson, 1997).

In general, for a theory to have scientific value it should deal with intransitive and relatively long-lasting domains of research; however, the question is: do intransitive social phenomena of long duration exsist? Bhaskar claims that they do, by arguing that while neither individuals nor groups fulfil the duration criterion, relationships do. According to Bhaskar, social sciences should primarily deal with lasting relations between individuals/groups, relations between such relations, relations to nature and the products of such relations (Buch-Hansen and Nielsen, 2007).

In social science, the much debated question of how the relationship between actor and structure should be conceptualized is illustrated by Buch-Hansen and Nielsen (2007) by the example of a phenomenon such a criminality; should the cause be found in the criminal or the context in which he exists? Critical realism maintains the traditional contrast between actor and structure, but instead of trying to eliminate the dualism by introducing new concepts (as in many other theories), the focus is placed on the interplay between actors and structures over time. It is argued that social structures always exists prior to human activities, which, on the other hand, always play a part in recreating or changing these structures. Hereby, it is recognized that humans/groups have a certain degree of freedom to form their own destiny, while they are also being restrained by their context (Buch-Hansen and Nielsen, 2007).

Methodology

Critical realism represents a wide-ranging methodology that makes positivism and radical social constructivism seem like two special cases at each end of its methodological space. Critical realism breaks away from both the perception 'that all knowledge is relative' and the perception 'that our knowledge is limited to what can be extracted from a hypothetical-deductive approach'[22] (Jespersen, 2005). In its methodological approach, critical realism emphasizes that it is the independent reality that is to be understood and explained, while recognizing the fallibility of all knowledge. Critical realism warns against the consequences of looking at problems with a one-sided theoretical perspective, but advocates the importance of interdisciplinary approaches across areas of natural science, social science and economics (Buch-Hansen and Nielsen, 2007).

In critical realism, the ontology of the domain that is researched is central, and therefore the methodical praxis should emerge from here. This means that the basis for research is a characteristic of the domain being researched, which forms the frame of reference for the research task. Critical realism emphasizes the importance of explicitly relating research to the existing knowledge, both positively and negatively, and that there is no justified knowledge basis that allows this task to be avoided (Buch-Hansen and Nielsen, 2007).

Having argued that the choice of research method should be reflected by the domain being researched, the question is how to choose such appropriate methods. Critical realism is pluralistic, but not relativistic when it comes to methodology. Therefore, although different methodological approaches apply to the same problem, the choice of methods is not free (Buch-Hansen and Nielsen, 2007). While it is not possible to say anything absolute about the 'correct' choice of method, the overall consequence of the methodological reflections of Bhaskar and Lawson can be formulated as (trans. from Jespersen, 2005):

1) "What are we looking at?" (characteristic structures of the researched domain, evaluated against the problem statement)
2) "What knowledge can possibly be acquired?" (epistemology)
3) "How can we acquire this knowledge?" (analytical approach)

The movement advocated by critical realism, from the empirical domain to the actual domain to the real domain, poses a challenge, which in critical realism is often attacked by the use of 'retroduction' and 'abstraction'. In brief, in critical realism, retroduction is an inference, which with basis in a manifest phenomenon or a given action, uses reason to arrive at which possible conditions and

[22] Hypothetical-deductive method: 1) define hypothesis; 2) generate predications from the hypothesis; 3) use experiments or observations to test predictions; 4) make conclusions.

causalities must exist for the phenomenon or action to take place. Abstraction is used in the meaning of the process of thought experiments that isolates certain aspects of a concrete phenomenon (Buch-Hansen and Nielsen, 2007). But critical realism also advocates the use of more traditional ways of reasoning, i.e. induction, deduction, and abduction,[23] where induction may be combined with deduction in a way that these two principles are complementary (Jespersen, 2005).

Another methodological core perception of critical realism is 'clarity of concepts'. According to Buch-Hansen and Nielsen (2007), Bhaskar basically argues that while measurements and quantitative methods make perfect sense in natural science, social science is subjected to clear limitations in that objects are based on meaning and concepts. Therefore, he suggests that concept clarity be given the same status in social science as exact measurements are given in natural science.

5.4.2 Discussion of critical realism

As mentioned, my main initial scientific theoretical problem at the beginning of this PhD project was that I was reluctant to drop the idea that knowledge creation is dependant on the context (in opposition to positivistic notions), while on the other hand, I would not disregard the idea of an independent natural and social reality to be studied (in opposition to social constructivism). Critical realism represents exactly this view of science and provides a solution to the problem of ending in relativism when acknowledging that the construction of knowledge is dependent on context, namely by arguing for our ability to apply rational judgement and that the objective standard, which socially constructed knowledge should be measured against, is the reality. Obviously, this may be difficult to implement in praxis, but compared to the 'context ignorant' or 'relativistic' alternatives, it seems like a more fruitful approach. There are also several other points in critical realism that make good sense in my view.

First, I find the idea of critical realism of mechanisms, causal potentials and tendencies to offer a good explanation model, also in configuration research. To give a simplified example: PVMs may have the causal power to represent some product knowledge, but has to be activated under the right circumstances for an expected effect to emerge. From here the focus of the research must be to identify these circumstances, e.g. qualifications of users, complexity of product and tools available. In the same way some particular conditions must be fulfilled for a knowledge engineer to be able to create knowledge representations. Here the causal potential of the knowledge engineer to represent product knowledge may be neutralised by expressional limitation of a chosen notation technique, the complexity of the products on focus etc.

Another point of critical realism that corresponds with my perception of science is the controlled methodological pluralism of critical realism. I often come across research where it seems that particular methods are uncritically applied, both in positivistically based research that involves social aspects and in social research. Positivistic research that deals with areas where social factors are of importance, but which insists on expressing all problems and solutions mathematically and quantitatively, consciously ignores parts of reality, and therefore, sometimes arrives at rather bizarre generalizations. On the other hand, I have also encountered socially oriented research that avoids quantitative measures and experiments, although such seemingly could provide valuable insights. When dealing with areas where social factors are significant, a typical reaction to the problem of fitting theories of science to the research being carried out is to neglect to reflect about choice of methodology and thereby reduce theory of science to methods. However, research with perfectly described and applied methods, but without any reflections of their value for investigating a given problem, bears great risk of blind-spots and imperfect conclusions. I find it to be a significant strength of critical realism that such methodological reflections are essential, while for obvious reasons this task cannot be put into formula. The pluralistic perspective on choice of methods in critical realism could be interpreted as a pragmatic way of choosing research methodology, but I see it as the opposite. Critical analysis of existing knowledge and reflection on choice of methodology require far more effort than uncritically applying the normally used methods from the field of research one belongs to. At a more practical level, I see a great strength in the ideas of retroduction and abstraction when investigating areas where social aspects are an important factor.

[23] Deduction: rule -> case -> result; Induction: case -> result -> rule; Abduction: rule -> result -> case

I believe that a focus on the actor-structure perspective of critical realism could be useful in many contexts to help avoid static or one-dimensional explanations of phenomena. Also, Bhaskar advocates a focus on relationships when dealing with social contexts, which is much in line with a systems theoretical perspective.

Finally, I subscribe to the argument of the importance of clarity of terms and concepts, as advocated by Bhaskar. Ambiguous meaning of terms and concepts is a problem that I have often experienced when studying relevant literature. Such ambiguity makes the literature difficult to read and easy to misunderstand, and often leads to my disregarding such work. I therefore perceive clarification of the meanings of terms and concepts to be one of the most important tasks within configuration and mass customization research in order to ensure scientific progress.

5.4.3 Final reflections on open and closed systems

The research carried out in order to answer the seven research questions can be seen as having been conducted in both open and artificially closed systems. Therefore, before describing the applied research methods, the implications of the type of systems investigated are discussed.

Much of the research carried out in this thesis deals with graphical notations for representation of product and configuration knowledge. When designing or modifying such graphical notations, it is required that the new or modified notations in some way contribute some benefits in comparison to the existing ones. Therefore, comparisons with existing notations are essential. The part of such comparisons that relates to what a notation is capable of expressing can be carried out in artificially closed systems. Here, the term 'artificial' refers to the fact that it is not a situation from practice but a constructed situation, which allows a controlled experiment to be carried out. The closure can be achieved by assuming that a modelling technique consists only of the elements defined, although in the open system of practice, additional modelling concepts may be invented while using this technique. Compared to analysing open systems, the experiments in the artificial closed systems allow many absolute conclusions to be drawn, i.e. fully verified claims. Such verification can be done based on logical argumentation or by pointing to facts of closed completely defined systems; e.g. the PVM formalism includes two relationship types, while class diagrams include these two and more relationship types; therefore, class diagrams allow expression of 'this and this' aspect of reality, while PVMs do not. But, as mentioned, in practice it cannot be presumed that users will comply with the defined formalisms but may invent variations of the formalism in order to cope with problematic modelling aspects. Furthermore, although in the defined closed system we show that one notation in some way works better than another, experience from practice may show that the situation represented by the defined problem may not be that relevant, or that an unforeseen problem emerges from the notation that worked best in the closed system.

Not all relevant aspects in the research of this PhD project can be investigated in closed systems, since many aspects require user involvement. In such open socio-technical systems, unknown circumstances can affect a given result. As a consequence, predictions of the course of events in such contexts cannot be generalized in an absolute fashion, but we can only talk about tendencies or good reasons to believe.

Chapter 6: Presentation of papers

In this chapter, the papers included in the PhD are summarized. But first an overview of the papers and how they deal with the research questions is presented.

6.1 Overview of the papers

During the course of my PhD project, I have produced 15 conference/journal papers as first author and co-authored two additional papers. A list of my publications is found in appendix 2. Of the 15 first-author papers, nine have been selected for this dissertation. This selection has been made in order to sort out papers that lie outside the main focus and to avoid overlaps. Table 1 presents an overview of the papers appended in this thesis. Appendix 3 contains a description of my contribution to these papers.

No.	Title	Year
A) Domain expert knowledge		
A1	Tacit knowledge in configuration projects	2007
A2	A classification of the information that domain experts do and do not provide in configuration projects	2007
B) Knowledge representation techniques		
B1	Product analysis as a basis for building product configuration systems	2005
B2	A comparative study of two graphical notations for the development of product configuration systems	2007
B3	Product structured class diagrams to support the development of product configuration systems	2007
B4	Merging models with different perspectives on product configuration knowledge	2006
C) Documentation of configuration knowledge		
C1	The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems	2007
C2	CRC-cards for the development and maintenance of product configuration systems	2006
C3	Creating a documentation system to support the development and maintenance of product configuration systems	2007

Table 1: Included papers

As can be seen, the nine papers are not organized according to chronology but according to topic. Therefore, when reading the papers, some inconsistency may be experienced regarding terminology, but not regarding the central points.

Table 2 presents an overview of which papers contain answers to the research questions of this thesis. The crosses in brackets indicate that a given paper is in some way relevant for the given question, although the question is not the main focus of the paper.

Question/Paper	A1	A2	B1	B2	B3	B4	C1	C2	C3
1) Does the use of the term 'tacit knowledge' in configuration literature comply with the original meaning of the term, and does it make sense to apply this term in configuration research?	X	(X)							
2) Can the knowledge/information that a domain expert possesses and delivers to a knowledge engineer in a configuration project be categorized in a better way than the tacit-explicit knowledge distinction?	(X)	X							
3) What are the limitations of applying the PVM formalism, and how can the formalism be altered in order to solve such limitations?			X	(X)	(X)		(X)		
4) What are the actual differences between using PVMs and class diagrams for modelling problems in configuration projects?				X	(X)				
5) How can the migration of information from PVMs to class diagrams be avoided while not loosing the benefits of the application of both techniques?					X		(X)		(X)
6) How can models with overlapping information in configuration projects be maintained, while avoiding having to update the same information in several places?						X	(X)		
7) What are the necessary definitions for the creation of a documentation system that supports the CPM-procedure?			(X)		(X)	(X)	X	X	X

Table 2: Answers to the research questions

Next, the included papers are described in relation to their purposes, propositions, discussions, research methods, results and conclusions. The summaries provide more extensive descriptions of the applied research methods than are found in the papers, including the basic reflections that led to the choice of methods. In this connection, it should be pointed out that my subscription to critical realism should only be seen as a subscription to an overall scientific frame, within which my research is carried out. My subscription to critical realism is therefore at a somewhat general level, concerning such central aspects as: ontological realism; context dependency and fallibility of knowledge; rational judgement instead of relativism; and that the choice of research methods should be based on the ontology of the domain that is studied. Critical realism terminology does not imbue my research; in fact, such terminology is not included in the appended papers. This is mainly because the focuses of the papers and the audience for whom they are written do not urge that the scientific aspects are described on much more than a methodological level.

6.2 A) Domain expert knowledge

6.2.1 Paper: A1

Title: Tacit knowledge in configuration projects

Purpose:

When developing product configurators, a major challenge is to represent relevant domain knowledge, since much of this resides in the minds of domain experts. In configuration literature, the term 'tacit knowledge' is often applied to describe some sort of non-explicit knowledge that is difficult to deal with. However, many different views exist on what tacit knowledge is, and since the existing literature on product configuration seldom provides any definitions or concrete examples of tacit knowledge, it is unclear what the term refers to in this literature. This lack of clarity means that it can be difficult to

build on existing literature. Therefore, this paper clarifies the meaning of the term 'tacit knowledge' and investigates the usefulness of the term in configuration research.

Research method:

To start with, configuration literature was investigated in order to describe its use of the term 'tacit knowledge'. Since the existing configuration literature found did not provide definitions of how the term is applied, the challenge was to: derive what kind of meaning it is attributed; compare this derived meaning with its original meaning; and conclude which meaning would make the most sense when being applied in configuration literature. To do this, an inductive-deductive kind of reasoning was applied. From all the found instances where the term 'tacit knowledge' has been used in configuration literature, induction was used to produce a conditional generalization of how the term is being applied, i.e. which meanings the term is attributed. The implication of the uses of the term 'tacit knowledge' in configuration literature is illustrated in figure 19.

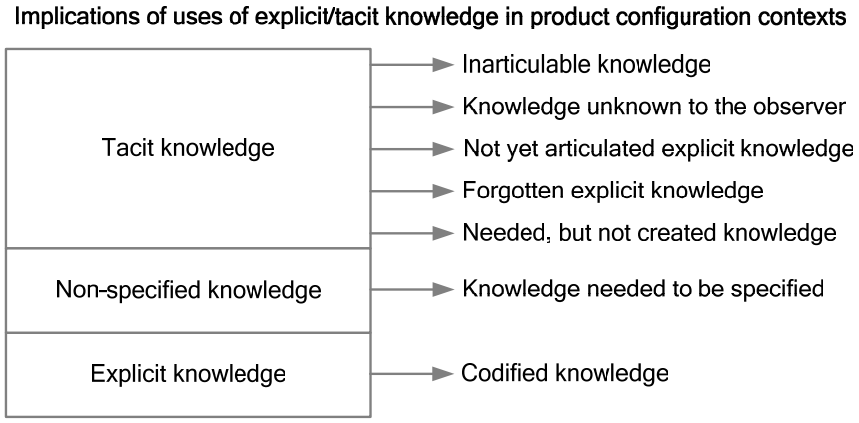


Figure 19: Implications of the uses of the term 'tacit knowledge' in configuration literature

From these findings, it could be established that the way the investigated configuration literature applies the term makes it ambiguous and uninformative, by reducing in principle the meaning of the term 'tacit knowledge' to 'knowledge or information that is not known to an observer'.

Next, the original definition of the term was applied to the contexts that in configuration literature are claimed to be a tacit knowledge phenomena (i.e. deduction), which showed that only few of these contexts in fact involve 'real' tacit knowledge.

Conclusions:

The paper argues that in a product configuration context it would make most sense to apply the term 'tacit knowledge' only about inarticulable knowledge. This definition is close to the philosophical roots of the term, and it avoids the definition of tacit knowledge being so broad that it loses its relevance by not telling much about the nature of the knowledge in focus. But by applying a strict definition of 'tacit knowledge', it seems that tacit knowledge is not particularly relevant when describing the difficulties of representing domain knowledge in configuration projects. The sharp line between what should be classified as tacit knowledge and what should not, implies that the distinction between tacit and explicit knowledge may not be a useful way of dividing knowledge in a configuration context, since the fact that some knowledge is tacit does not necessarily represent a difficulty in regard to creating a configurator, just as explicit knowledge by definition cannot be considered to be easy to convert into a model in the knowledge base of a product configurator. Finally the paper argues that tacit and explicit knowledge should be seen as two very different concepts, and not two ends of a continuum. Therefore, tacit knowledge cannot be converted into explicit knowledge or the other way around. What actually happens, when it is claimed that tacit knowledge is made explicit, is that new explicit knowledge is created, based on what can be inferred by observing the tacit knowledge in action.

6.2.2 Paper: A2

Title: A classification of the information that domain experts do and do not provide in configuration projects

Purpose:

This paper proposes a classification of the information that domain experts do and do not deliver to knowledge engineers in configuration projects. The classification represents an alternative to the tacit-explicit distinction that is applied in configuration literature to describe problematic and unproblematic knowledge, respectively, but which contradicts the original meaning of the term 'tacit knowledge' and can be misleading, as argued in the preceding paper (A1).

Proposition:

The paper proposes the following classification of the information that domain experts do or do not provide to a knowledge engineer in knowledge elicitation situations:

- Information that does not leave a domain expert:
    1) Concealed information
    2) Unrecognized information
    3) Non-possessed information
- Information that is not usable:
    4) Incorrect information
    5) Irrelevant information
- Information that requires analysis:
    6) Inarticulable information
    7) Contradicting information
- Directly usable information:
    8) Relevant explicit information

There are two arguments for using the term 'information' instead of 'knowledge'. First, this avoids the discussion of whether 'pure explicit knowledge' actually exists, since some would argue that even the most explicit form of knowledge is underlain by tacit knowledge. Second, traditional definitions of knowledge include a 'truth' element, which makes little sense when talking about e.g. incorrect and contradicting statements.

Research method:

Being based only on the literature and my personal experience, the relevance and validity of the proposed classification was investigated by studying four configuration projects. The studies were carried out through interviews with some of those who are most knowledgeable about which kinds of information emerge in configuration projects, namely knowledge engineers. They were given the task of trying to falsify the hypothesis, 'the classification covers all kinds of information that emerges in the defined context', by providing examples of types of information that did not fall into any of the categories. For the classification to be relevant, the knowledge engineers should also provide verification that the defined categories of information were all relevant. To ensure that the interviewed knowledge engineers did not give not-thought-out answers or misunderstood the meaning of the defined categories of information, they were asked to provide concrete examples of situations where the proposed categories of information had emerged in the projects in which they had participated.

The interviews were carried out as semi-structured interviews.[24] A separate part of the interview where it was asked whether each of the defined categories of information had been encountered and

[24] Fully structured interview: "has predetermined questions with fixed wording, usually in a pre-set order. The use of mainly open-response questions is the only essential difference from an interview-based survey questionnaire." (Robson, 2002)

how often (closed question[25]/scale item)[26] can be perceived as a 'fully structured interview'.[27] In the part of the interview about the situations where they had encountered different categories of information, it was necessary to be able to adapt the questions (open questions[28]) to the answers given during the interview. This part of the interview corresponds to an 'unstructured interview'.[29]

Results:

All the interviewed knowledge engineers agreed that the proposed classification was extensive enough to cover all the categories of information that domain experts do and do not provide in a configuration project. Therefore, the test did not lead to falsification of the hypothesis concerning the extensiveness of the proposed classification, implying that this hypothesis can be considered so far to show a tendency towards being true. In addition, the interviews supported the claim that all the defined categories were relevant, since concrete situations where all categories of information had emerged were described by the interviewed knowledge engineers. This verification is obviously connected with uncertainty, since it is based on subjective judgements, and can therefore not be perceived as a basis for a very definitive conclusion.

According to the interviewed knowledge engineers, inarticulable information (or tacit knowledge) played an insignificant role. Instead, the investigations showed that the real challenges in product configuration projects seem to be: how to create not yet defined information (non-possessed information); how to make domain experts agree on what information to use (neutralize contradicting information); and how to decide what is relevant to include in a model (avoiding irrelevant information).

Conclusions:

By presenting seven kinds of information that could represent problems in a knowledge acquisition situation, this paper provides a much more nuanced basis for future configuration research. In addition, the classification may be of use to other areas of research that deal with situations where information is elicited from domain experts.

6.3 B) Knowledge representation techniques

6.3.1 Paper: B1

Title: Product analysis as a basis for building product configuration systems

Purpose:

This paper deals with two aspects of product configuration. The first comprises a discussion of how the product perception from the field of 'engineering design' (theories of technical systems), which CPM builds upon, corresponds with a configuration context. Based on analysis of the strengths and weaknesses of the existing PVM formalism, the second aspect entails the proposal of a new graphical notation technique for describing products in the analysis phase of configuration projects.

[25] Closed questions force the interviewees to choose from a fixed set of alternatives.

[26] Scale items are an answer type with of degree of agreement/disagreement or similar judgements. Scale items are logically of the closed question/fixed-alternative type, but sometimes regarded as a separate type (Robson, 2002).

[27] Semi-structured interview: "Has predetermined questions, but can be modified based upon the interviewer's perception of what seems most appropriate. Question wording can be changed and explanations given; particular questions which seem inappropriate with a particular interviewee can be omitted, or additional ones included." (Robson, 2002)

[28] Open questions do not have restrictions on the content of the answer, other than on the subject area.

[29] Unstructured interviews: "The interviewer has a general area of interest and concern, but lets the conversation develop within this area. It can be completely informal." (Robson, 2002)

Discussion of concepts:

The analysis of engineering design theory and its position in a configuration context led to the proposal of a revised version of the applied variant of the 'chromosome model' that is prescribed in CPM literature for organizing a PVM model (Riis, 2003). In the proposed redefinition, the levels are changed from: function, organ/property, and part/property models to: property/organ, component, and life-phase models. The change consisting of using properties instead of functions (which is a type of property) at the top level can be justified by the argument that the superior function of a product is often very obvious (e.g. 'allows persons to sit in' for a chair) and therefore does not need to be explicitly described in a configuration context, while other superior properties do (e.g. the price of a chair).

Research method:

The question of which shortcomings and strengths the PVM technique possesses was investigated by applying a kind of inductive-deductive approach. First, empirical studies were carried out in order to generalize which problems emerge in practice when applying the PVM technique (induction). Next, a new notation designed to solve these problems was proposed. For this new diagram to solve the identified problems of PVMs, a set of problem types should show that this is the case. Therefore, such problems were defined and the proposed notation was submitted to these problems (deduction).

The empirical studies include three elements: 1) study of existing research data, 2) interviews, and 3) observations. The existing research data was in the form of a report, sound-files and transcripts of interviews. The advantage of using existing data was obviously that it saved a lot of work preparing and carrying out the interviews; on the other hand, I had no influence on the questions asked and was left with unanswered questions. Based on this overview, interviews with seven persons from four of these projects were conducted, i.e. persons with experience from modelling with the PVM technique. The purpose of the interviews was only to gain information about the experiences of the users, and therefore no or only few questions were prepared in advance for the interviews (i.e. unstructured and semi-structured interviews). Finally, observations of five modelling sessions in ongoing projects were made and followed by brief interviews. The modelling sessions involved knowledge engineers and domain experts who discussed and refined their PVM models. The use of observations allowed me to observe what people were in fact doing, rather than hearing their subjective opinions, which do not necessarily comply with practice.

Four dimensions of participation can be distinguished when conducting observations: the complete participant;[30] the participant as observer;[31] the marginal participant;[32] and the observer-as-participant[33] (Robson, 2002). Two of the observations carried out can be classified as 'participant as observer' and the last three as 'marginal participant'. The first two observations were from a project where I myself was involved in the creation of conceptual models, but the model in focus was to be created by another knowledge engineer. This position meant that although I was not the one asking the questions or drawing the model, I was drawn into the discussion and made suggestions, i.e. considered as a participant, although I had made it known that my focus in these sessions was observation. The last three observations involved a case where I was allowed to observe some of the modelling sessions in a project in which I was not otherwise involved. The study was intended to be carried out as 'observer-as-participant', but because of my status as an expert, I was occasionally drawn into the modelling sessions, and thereby became a 'marginal participant'. The major concern of the observations is obviously how my presence affected the participants in the modelling sessions; i.e., did I unconsciously draw the sessions in directions that would create problems? I believe that the relative simple focus of the observations, and my being aware of this danger, minimized the chance of this.

[30] The complete participant: the observer conceals that he/she is an observer and observes through participating in relevant activities (Robson, 2002).

[31] The participant as observer: it is made clear from the start that the observer is an observer, and based on this, the observer participates in relevant activities (Robson, 2002).

[32] The marginal participant: the observer has lower degree of participation than the preceding two kinds of observation (Robson, 2002).

[33] The observer-as-participant: the observer takes no part in the activity, while the status as a researcher is known to the participants (Robson, 2002).

Finally, the subsequent interviews allowed me to ask the participants why in certain situations they made certain choices regarding the chosen way of representing specific aspects; i.e., I could investigate whether they interpreted the PVM formalism differently than I do and what their arguments were for inventing specific new formalisms during the modelling sessions.

Results:

The studies of the company projects revealed needs for: 1) extended notation formalism; 2) solution of inexpediencies when using kind-of structure; 3) better possibilities for graphically displaying constraints on relations; 4) preparation of the notation for IT-supported modelling; and 5) increased utility and instruction in applying notation for modelling the product through life-phase systems.

Proposition:

The problems related to expressional limitations of the PVM technique were sought solved by defining a new notation based on the PVM notation, named Product Family Diagrams (PFDs). The PFD formalism is shown in figure 20. Also, the paper proposes a variant of the PFD notation to be used for modelling of life-phase systems.

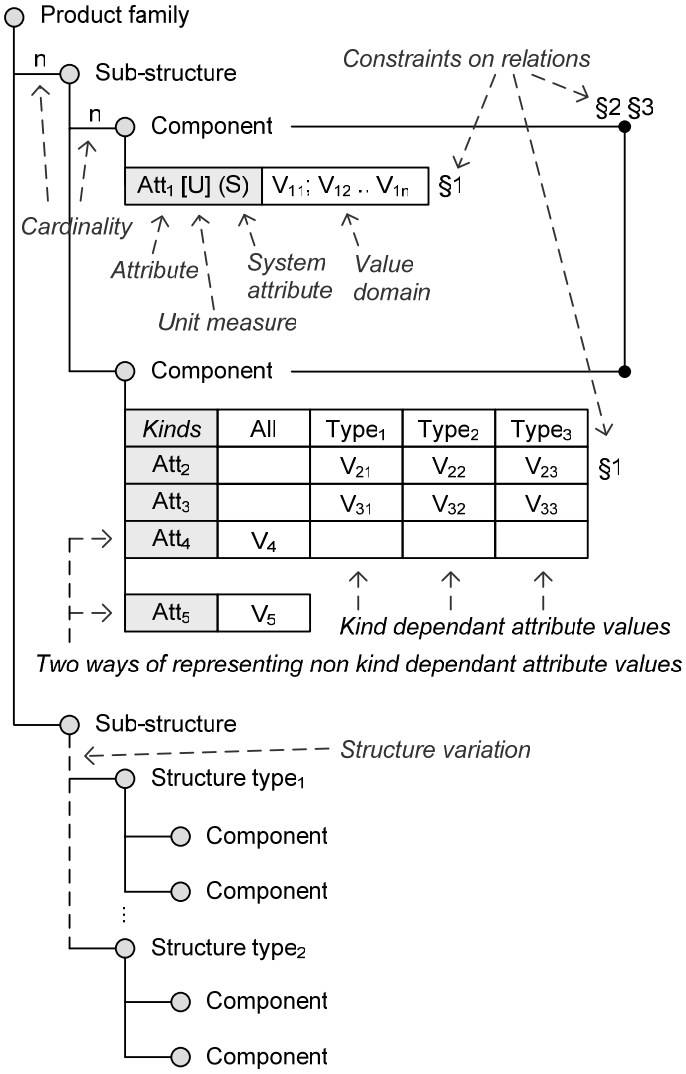


Figure 20: The PFD formalism

Research method (2):

To test if the new notation actually solved the found problems, principal modelling problems were modelled using PVMs and PFDs. These experiments showed that in the specific contexts, PFDs solved the problems of the PVM notation.

Conclusions:

While it is possible to conclude that PFDs solve the problems of the PVM notation for certain modelling tasks, it must be recognized that it may create other problems for other types of modelling tasks. Therefore, in order to say something more conclusive, more experiments would have to be carried out. Furthermore, the great shortcoming of the tests of the new notation is that the usability dimension was not investigated, although this may be the most significant factor. This does not disqualify this research, but as recognized in the conclusion of the paper, the notation is far from being adequately tested in a way that allows concluding that it is a useful alternative to the PVM technique in all cases.

End comment:

This paper was only my second, which I believe is apparent when comparing it to later work. Also, in retrospect, I agree with the recommendation of Professor Mogens Myrup Andreasen, who argued that the paper should have been divided into two separate papers: one dealing with the division of a configuration domain, and the second with a new graphical notation technique.

6.3.2 Paper: B2

Title: A comparative study of two graphical notations for the development of product configuration systems

Purpose:

The purpose of the paper is to compare the PVM and class diagram notations. These two graphical notations are part of the CPM-procedure and have in configuration projects been applied both in extension of each other and alone. This obviously raises the question of when to use both or only one of these. In CPM literature, it is claimed that the PVM notation is more user-friendly, while class diagrams provide a richer and more formal language. However, such assumptions have never really been investigated nor has the reason for this difference been explained. Therefore, such claims may be hard to communicate. It can also be imagined that by providing explanations for why PVMs are more user-friendly and class diagrams richer and more formal, alterations of these notations or the invention of new ones could provide an even better basis for future configuration projects.

Research method:

The assumption of CPM literature, which maintains that PVMs are user-friendlier when extracting knowledge from domain experts, whereas class diagrams are better suited for making formal definitions of the knowledge base of a configurator, was investigated by comparing the expressional strengths and usability aspects of PVMs and class diagrams. The analysis of expressional strength was made by comparing the two notations in relation to possible modelling scenarios; the usability comparison was based on studies of product configuration projects. The comparison of the expressional strengths was based on facts of the two notation techniques. Thus, it could be argued that one notation technique may provide certain advantages in specific configuration contexts, because it includes specific modelling constructs or offers other modelling possibilities not included in the other notation technique. Much more difficult was the investigation of the usability of the two notations. This investigation included: four cases, studied by conducting un-/semi-structured interviews with eleven people from these companies; reviews of earlier studies; analysis of documentation produced by the companies; and observations of the use of the PVM notation in two projects. The observations and studies of earlier research correspond to those described in paper B1.

Results:

The comparison of the expressional strength of the two notation techniques showed several limitations of PVMs compared to class diagrams, whereas no real advantages appeared in this respect. These limitations include: the limited range of modelling concepts; not allowing the modelling of multiple parents; inflexibility when it comes to creating interrelated models; the low degree of formalism; and the lack of software support for the notation. On the other hand, all the interviewed persons of the studied companies were of the opinion that PVMs are easier to learn and work better in knowledge elicitation situations that include domain experts.

Discussion:

Instead of just accepting the subjective experiences of the interviewed knowledge engineers that PVMs are easier to learn and overview than class diagrams, what can be considered a retroductive analysis was carried out. The PVM technique seems in general to be considered user-friendlier than class diagrams (recognizing the limited amount of research data). The perception of PVMs in the companies studied can be explained by the fact that some of the symbols and ways of organizing them in PVM models correspond to the way in which relevant persons are accustomed to representing this information, for instance in BOMs, indexes, software etc. This explanation supported the empirical data, which means that a stronger conclusion can be made.

Conclusions:

The study of the expressional strength of PVMs and class diagrams showed that class diagrams have many advantages over PVMs, whereas the PVM technique has no such advantages. On the other hand, based on the studies carried out, it seems that it is easier to learn the PVM notation than that of class diagrams for persons without or with only limited modelling experience.

The paper argues that the apparent relatively easier learning of PVMs can be explained by two main PVM characteristics: 1) that the way the relationship types are represented resembles to some degree well-known concepts, contrary to the symbols of class diagrams; and 2) that the different relationship types are placed in different columns, contrary to normally drawn class diagrams where these are mixed together. The placement rules for PVM elements implies a reading pattern that resembles that of text; therefore, PVM models (also only partly constructed) seem to be better suited for reviewing than class diagram models.

Finally, the paper suggests that placement rules in PVMs may also be possible to apply to class diagrams as well, thus giving class diagrams comparable usability to PVMs. If this is the case, the argument for choosing PVMs instead of class diagrams would be reduced to preference for special symbols to represent different concepts.

6.3.3 Paper: B3

Title: Product Structured Class Diagrams to support the development of Product Configuration Systems

Purpose:

The CPM-procedure prescribes the use of PVMs for product analysis and class diagrams for the design of the configurator knowledge base. This use of diagrams implies that information has to be transferred from one diagram type to another, which can be time-consuming and lead to errors. To deal with such problems, the paper builds on the proposition from the preceding paper (B2) to apply the placement rules of PVMs to class diagrams in order to obtain the advantages of both these diagram types at the same time. If this layout principle for class diagrams maintains the advantages of both diagram types, it could eliminate the need for the transfer of information from PVM models to class diagram models.

Proposition:

The paper defines how the application of the PVM placement rules to class diagram can be implemented. This class diagram layout principle is named 'Product Structured Class Diagrams' (PSCDs). The principle is illustrated by an example in figure 21.

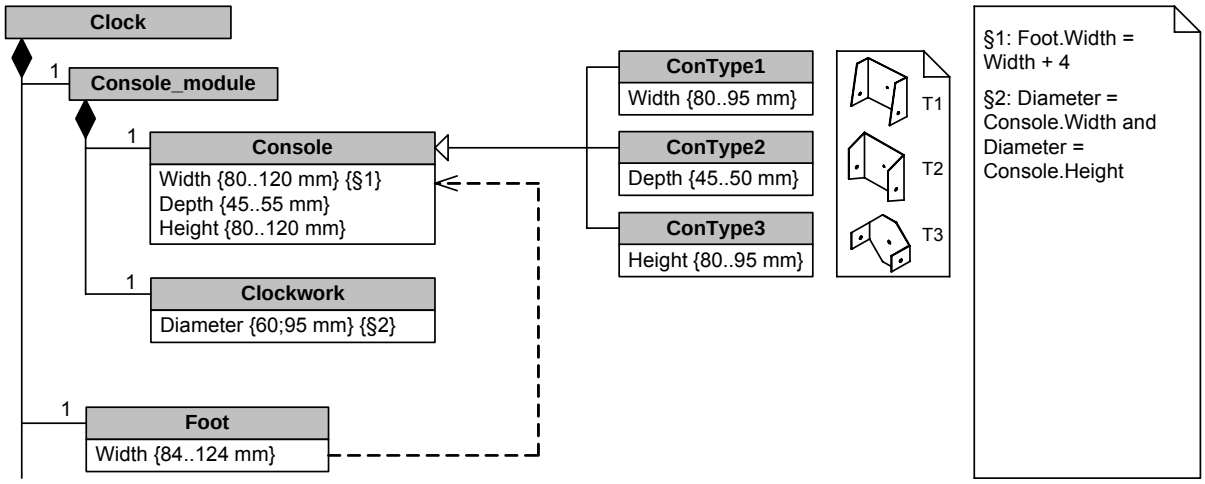


Figure 21: Example of the use of PSCDs

However, transplanting principles of PVMs to class diagrams could mean that some of the advantages of class diagrams would be lost. Furthermore, the explanation of why PVMs seem user-friendlier than class diagrams is only an open theory, so it cannot be concluded that the application of the relevant PVM characteristics would result in achieving the same usability as PVMs. Therefore, it was necessary to investigate the expressional strength and usability of PSCDs in comparison to class diagrams and PVMs.

Research method:

First, based on the preceding paper (B2), nine advantages of using class diagrams compared to PVMs were defined in relation to expressional strength. Since the preceding paper shows that the expressional strength of class diagrams is higher than PVMs on all points, PSCDs only needed to be compared to class diagrams in this respect. This comparison was made by analysing for each advantage point of class diagrams whether this also held for PSCDs. For some of the points, this was a fact (e.g. the connection to other UML diagrams), while other points had to be examined by defining a principle modelling problem and applying the PSCD technique to this (e.g. if it is possible to model classes with multiple parents).

The usability analysis of PSCDs proved to be more challenging, since no experience from the use of PSCDs in configuration projects existed. Therefore, it was decided to carry out a usability experiment[34] in which 18 engineering students, split into 9 groups, were given a modelling assignment using each of the three notation techniques, i.e. 3 groups using each notation. In the experiment, on the basis of a textual description of a product family, the groups were given a fixed period of time to model as much of the product family as far as they could, with no expectation that any of the groups could complete the task. The amount of content the groups managed to complete, number of errors, and size of the representation were measured. In other words, the 'independent variable' of the experiment is the applied notation; the 'dependent variables' are the number of each type of model element drawn, the number of errors, and the size of the model; and the 'controlled variables' are the problem definition, time period, materials for the creation of models etc. The variance of two other variables was also sought minimized, namely the motivation and prerequisites of the students, which is

[34] Experimentation is a research strategy that involves: 1) "the assignment of participants to different conditions"; 2) "manipulation of one or more variables (called 'independent variables') by the experimenter"; 3) "the measurement of the effects of this manipulation on one or more other variables (called 'dependent variables')"; and 4) "the control of all other variables" (Robson, 2002).

obviously very difficult. This is thus an important possible source of error that can lead to false conclusions if ignored.

Results:

It was shown that the implication of imposing the PVM placement rules on class diagrams was not significantly increased expressional limitations compared to the use of class diagrams without these restrictions. Doing the testing in well-defined closed systems allowed rather conclusive conclusions to be drawn, while the significance of the minor limitations that were considered unimportant is obviously debateable. The usability experiment, on the other hand, did not produce anywhere near as conclusive results. Although the measured dependence variables did indicate to some extent that PSCDs are easier to draw than class diagrams and are similar to using PVMs in this respect, the moderate variance of the dependence variables, the small sample size and possible sources of errors must be considered. A more dramatic difference than in the dependence variables can be found in the organization of model elements. Comparisons of the models drawn show that the model elements in the PSCD models are organized much more in vertical and horizontal lines than the class diagram models.

Conclusions:

It seems that PSCDs hold the expressional advantages of class diagrams, whereas the usability aspect is more uncertain. The small sample size, the significant possible sources of errors, and the lack of a dramatic difference in measured dependent variables imply that an adequate basis for making strong conclusions based on these data is not present. However, comparisons of the PVM, PSCD and class diagram models made by the students indicate that if a novice knowledge engineer is put under pressure, the use of PSCDs will probably result in representations that are easier to overview and review than the use of class diagrams, and with an effect similar to the use of PVMs.

6.3.4 Paper: B4

Title: Merging models with different perspectives on product configuration knowledge

Purpose:

This paper proposes a modelling principle to deal with a common issue in product configuration projects, but also in software development generally, namely the problem of maintaining models with overlapping content, which can be time-consuming and a possible source of errors. The paper goes into detail with two particular types of cases, namely cases where analysis and design models are needed, and cases where configuration is to be performed at different stages of a project by different types of domain experts. In addition, the modelling concept may also be applied to other types of modelling scenarios that include models with overlapping content.

Proposition:

The proposed solution is defined at two different levels, a modelling principle and definitions of how this principle could be supported by software. The basic idea of the modelling technique is to merge the models with overlapping content into one common model, while stereotyping classes, attributes, methods, constraints and relationships according to which model view they belong to. Furthermore, the proposed modelling principle includes definitions of how to handle classes, attributes, methods and constraints in cases where classes are represented as attributes in other model views. A definition of how to handle specialization classes, which in other model views should be merged with their generalization class, is also presented.

The modelling principle is illustrated in figure 22, where a common analysis and design model for a toy car is shown. Turning off the design model element layer (<<dm>>) produces an analysis model, and turning off the analysis model element layer (<<am>>) produces a design model. The stereotypes <<atc>> and <<cta>>, respectively, indicate if an attribute is converted to a class in another view and if a class is converted into an attribute in another view.

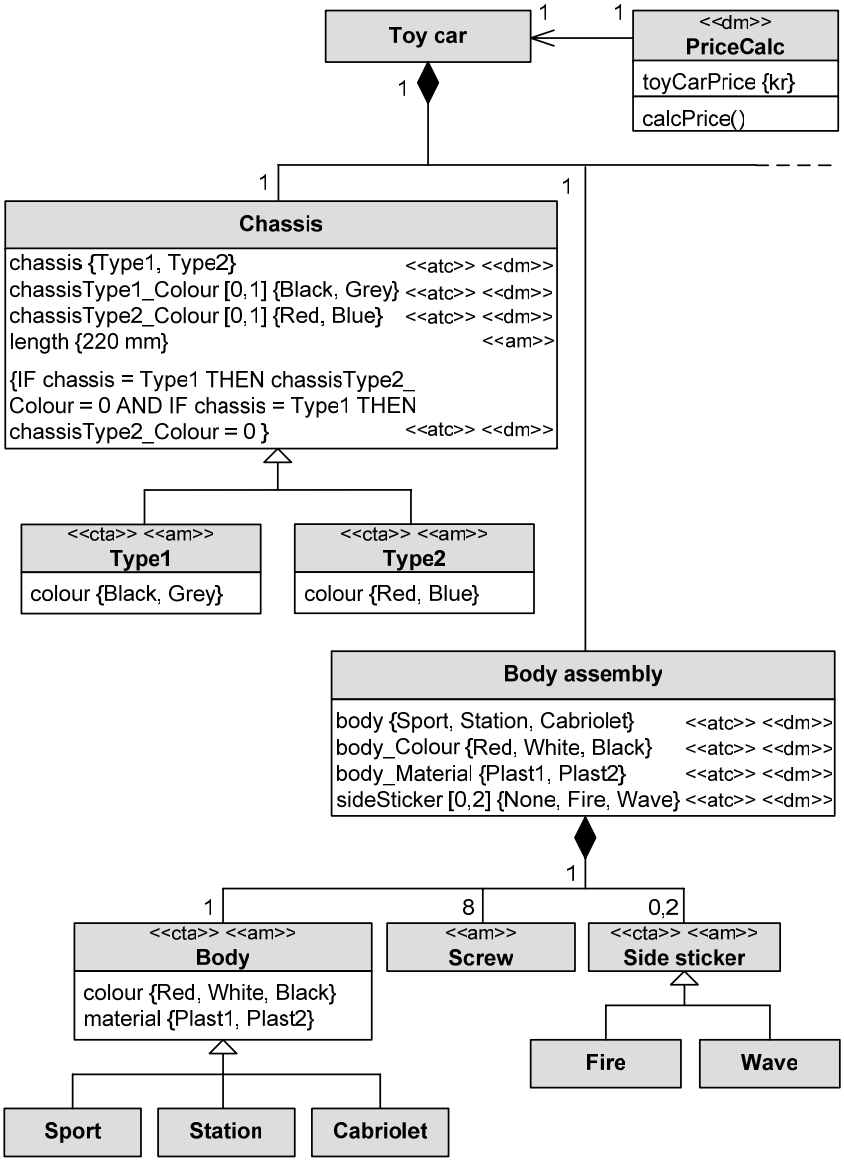


Figure 22: The total model of toy car

However, to be able to view different model views individually, software support is needed. Therefore, two possible software solutions are discussed in the paper. The first solution is based on MS Visio, where the layer functionality of Visio is applied to separate content belonging to different model views. Visio allows such layers to be turned on and off, so that different views can be produced by turning off model elements that do not belong to the desired model view. However, while this solution may work nicely in cases with little individual content for each model view, the solution has a major disadvantage in that the turned off model elements leave blank spaces in a model view. This solution also requires that the different model views must be created by using the same notation technique; i.e. it does not allow the elaboration of PVMs followed by class diagrams as prescribed in the CPM-procedure. The second proposed software solution is to include the modelling principle in a documentation system to support the development and maintenance of product configurators.[35] The software should generate different model views based on a common data model instead of a common graphic model, as in Visio. This implies that this kind of solution would support the creation of models by using different representation techniques. Also, because such software would dynamically generate

[35] Such a system has only been developed as a prototype so far, as later described in paper C3.

its model views based on the data model and placement rules, the problem of blank spaces in a model view would be eliminated.

Research method:

The proposed modelling principle was investigated through two kinds of tests. The first was to apply the modelling principle to a fictive modelling problem, while the second was to apply the proposed principle in practice. The latter was done in a configuration project, where I was working as a knowledge engineer, for which reason this can be perceived as a form of 'action research'.[36] Such a research method obviously presents some challenges in relation to bias, subjectivity and validity. To be more specific, it can be hard to know if the methods/techniques that are tested are really what solve a problem and not other actions by the researcher. However, in the current context, what was tested and the way it was to be evaluated were rather concrete, in that the main part of the test was done on product data and not humans. The common dangers of action research were therefore not very relevant in this case, and therefore relatively solid conclusions can be made about the applicability of the proposed modelling principle and the one technical solution.

Results:

The study of the application of (part of) the modelling concept in the configuration project showed that the modelling principle, together with the MS Visio, eliminated the need for separate models with overlapping content. This common model was used and elaborated throughout large parts of the project without negative comments from the domain experts that were to read the model. However, the modelling problem in the specific case was rather simple and only little content belonged to separate views. Thus, the conclusions to be made must consider this aspect.

Conclusions:

It has been shown that the proposed modelling principle has the constructs needed for merging different models with overlapping content in the two problems to which it has been applied. But obviously, more complex problems may show a need for additional constructs to control which views model elements belong to and are transformed from view to view. Experiences from application in an actual project showed that the use of the modelling principle supported by MS Visio worked, in the sense that two models with overlapping content could be merged and maintained in a common model, without a negative impact on their usefulness. However, it is recognized that in encountering more complex situations in which model views have much individual content, the Visio solution is not that suitable. Such situations would require a 'configuration documentation system'. A configuration documentation system is also a prerequisite, if different diagramming notations are used for the models to be merged.

6.4 C) Documentation of configuration knowledge

6.4.1 Paper: C1

Title: The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems

Purpose:

The paper deals with defining a documentation system that supports the three modelling techniques of the CPM-procedure: PVMs, class diagrams, and CRC-cards. The creation of a configuration documentation system has been a topic of research at CPM for some years. However, existing research from CPM focusing on the requirements for a documentation system does not provide detailed

[36] The term "action research" was first used by Kurt Lewin (Lewin, 1946) and describes "Research which is orientated towards bringing about change, often involving respondents in the process of investigation. Researchers are actively involved with the situation or phenomenon being studied." (Robson, 2002)

definitions of the notation formalisms and the mutual mapping of the modelling techniques that should be included. Creating a system without such definitions seems like a very difficult mission; therefore, this paper provides these definitions.

Discussion:

Existing research on documentation in configuration projects shows that the documentation task is often one of the first to be eliminated from a configuration project, and that such a decision can turn out to have very negative consequences, e.g. the inability to further develop a configurator. The paper suggests that the lack of documentation in many configuration projects can be explained by the fact that no software currently exists that supports the modelling techniques of the CPM-procedure in an integrated fashion. This can mean, for example, that in the development phase, MS Visio may be chosen for creating PVMs; Rational Rose for class diagrams; and MS Word for CRC-cards; and finally, an application like Lotus Notes in the maintenance phase. Such approaches require several manual transfers of information between models without automated checks for consistency across models. These transfers are time-consuming and possible sources of errors. For this reason, a software tool which supports the mentioned techniques in an integrated fashion could ease the documentation tasks considerably.

Research method:

As a basis for providing the needed definitions to create a documentation system, two of the most developed configuration documentation systems in operation in Denmark were analysed. User requirements were gathered through unstructured and semi-structured interviews, some as telephone interviews. Based on this, software design specifications were created. The elaborated specifications were continuously delivered to a student from the Department of Informatics and Mathematical Modelling at the Technical University of Denmark, who transformed them into a prototype (described in paper C3). The method for elaborating the system definitions was therefore mediation between the gathering of requirements from industry and fulfilling the need for the software specifications needed in order to create the system. The continuous evaluation of the software specifications by the programmer (i.e. the student) serves to ensure that these definitions provide an adequate basis for the creation of a documentation system. If this transformation of definitions into a software prototype had not been made, the software specifications would most likely have been inadequate.

Proposition:

For the configuration documentation system to support the modelling techniques of the CPM-procedure, the paper argues that the necessary definitions must at least include:

1) A formalization of PVM notation that allows software support
2) A definition of the subset of class diagrams to be included
3) An extended CRC-card definition
4) Definitions of how to handle relationships between PVMs, class diagrams and CRC-cards
5) A definition of which package diagram elements to include, and the relationship to class diagrams
6) Definitions of the basic windows of the software system

Four of the defined basic windows are shown in figure 23.

The redefinition/formalization of PVMs to be included in a documentation system includes a redefinition of symbols for expressing kind-of, attribute variance, cardinality and constraints/rules, so that these are easier to include in a software system and closer to the class diagram notation. Furthermore, a constraint sheet and stereotypes have been added to the notation formalism. In addition to what is defined in the latest CPM definitions,[37] the proposed definition of the subset of class diagrams to be included in the documentation system includes: dependency relationships, stereotypes, and more refined/normative rule expressions. The CRC-card navigation three (see figure 23) is a

[37] Hvam et al., 2007a.

variant of the PVM technique that holds both part-of and kind-of classes in the same column and excludes class information (attributes and constraint/rules). The definition of package diagram elements defines how to create relationships between separate models. The definitions of how to handle relationships between PVMs, class diagrams and CRC-cards are based on the use of stereotypes for handling: common components (reusable global models), classes that share CRC-cards, mapping from CRC-card navigation tree to class diagrams, and constraint classes.

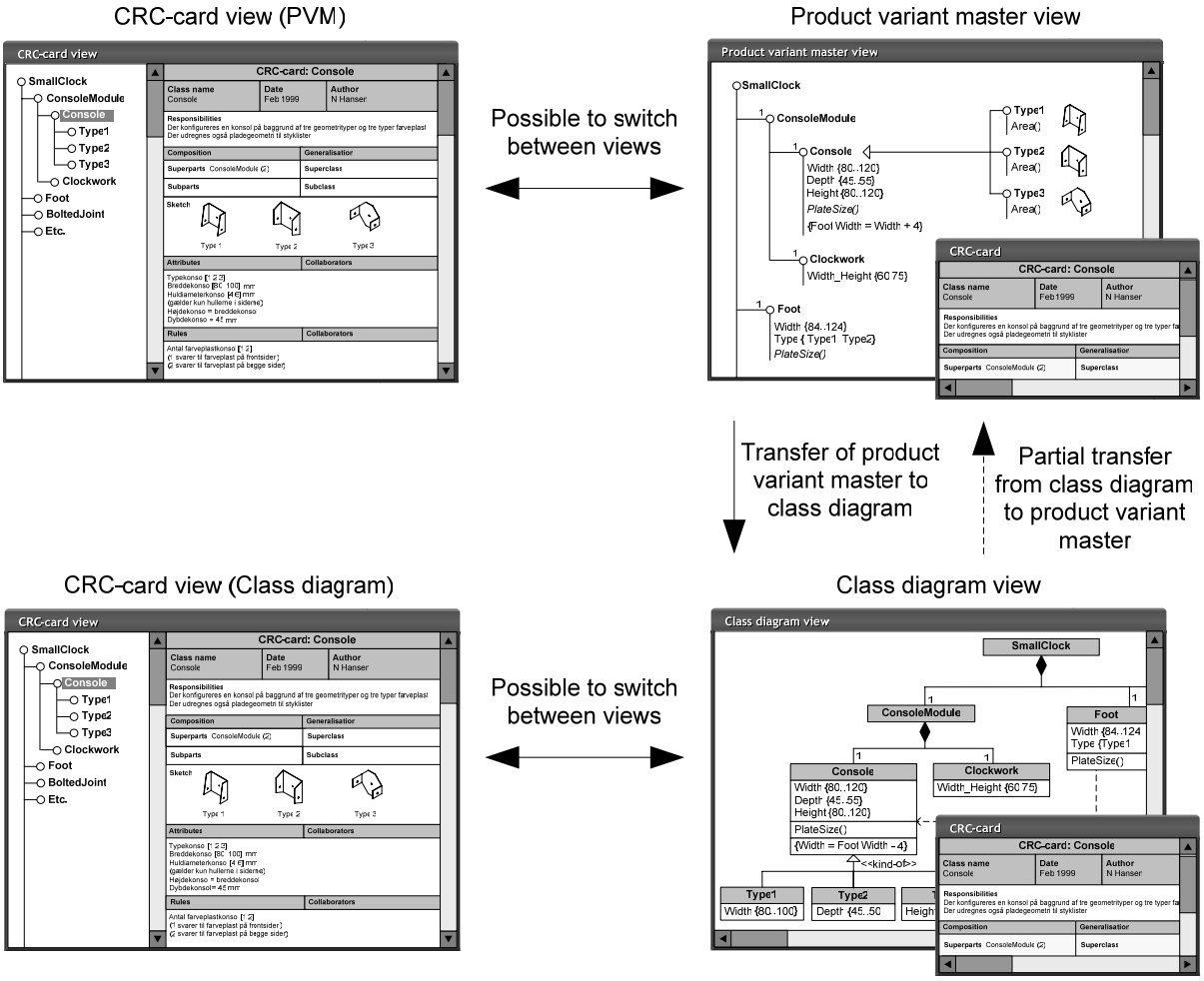


Figure 23: Overall documentation system design

Conclusions:

The paper has produced the definitions of a documentation system to support the CPM-procedure, and showed to some extent that the defined integration between modelling formalisms works. But obviously, to really investigate the definitions, a software prototype needs to be completed and evaluated. Also, the paper does not go into detail about the CRC-card definition for a documentation system, which is needed in order to support company-specific requirements.

6.4.2 Paper: C2

Title: CRC-cards for the development and maintenance of product configuration systems

Purpose:

Based on the fact that the CRC-card layout applied in companies does not resemble the CPM-definition and that further fields would be useful if CRC-cards are to be part of a configuration

documentation system, this paper presents a new definition of special CRC-cards to be used in the development and maintenance of product configurators.

Research method:

First, it was investigated how the CPM definition of the CRC-card layout has changed in the literature since being proposed. Next, the requirements for a CRC-card definition were acquired by conducting unstructured and semi-structured interviews with two of the companies with the most developed documentation systems in operation in Denmark, together with analysis of their existing documentation systems. Existing CRC-card definitions were then further developed by integrating the requirements of these companies and by taking other possible needs into account.

Results:

The studied cases show that CRC-cards can be a valuable technique to support the development and maintenance of configurators, but that the current CPM definition of the CRC-card layout differs markedly from the way in which CRC-cards are elaborated in praxis.

Proposition:

To provide a better basis for new companies adopting the CRC-card technique, the experience from the study was incorporated into a new CRC-card layout. Also incorporated into the new design were fields that are needed if the CRC-cards are to be part of a software-based documentation system that supports the development and maintenance of configurators.

The basic layout of the proposed CRC-card consists of the top level fields: Class (name), Status, Change requests (yes/no), and Version, together with six groups (folders) of class information: Basic information, Relationships, Sketch/Picture, Knowledge group, Change requests, and Change history. The group, named 'Knowledge group' is defined as a group that can have multiple instances according to the preferences of a specific company. In the case of one of the investigated companies, it would have the instances: Product knowledge, Price knowledge and Text knowledge.

The proposed CRC-card formalism is shown in figure 24, where the 'Basic information' folder is open, while the others are closed.

Class:		**Status:**	**Change requests:**	**Version:**
▷ **Basic information**				
Created by:	**Date:**	**Card responsible:**	**Product responsible:**	
Responsibilities:				
▶ **Relationships**				
▶ **Sketch/Picture**				
▶ **Knowledge group 1**				
⋮ ▶ **Knowledge group N**				
▶ **Change requests**				
▶ **Change history**				

Figure 24: Proposed CRC-card layout

Compared to the existing CRC-card definition from CPM, the layout proposed includes a long list of new fields, e.g. for documenting additional relationship types and organizing information about attributes, constraints and methods. Another important extension of the CRC-card layout is the groups for the handling change requests, which contrary to the other content of the proposed CRC-card layout requires software development to function efficiently.

The paper points out that the proposed CRC-card layout should be adapted from case to case to fulfil the needs of a specific company by including only the relevant fields of the proposed definition and by adding additional fields if necessary.

Conclusions:

The proposed CRC-card definition provides a flexible basis, which contrary to the existing CPM definitions could support the documentation work of the two investigated companies in its defined form. The paper argues that by having incorporated the experience gained from studies of two particular companies into a new CRC-card layout, an improved basis is provided for other companies taking up the technique. This new basis could translate itself into such benefits as minimizing redundant information and avoiding the neglect relevant aspects in the configurator documentation. Furthermore, the proposed definition of CRC-cards provides an improved basis for the creation of a documentation system to support the development and maintenance of PCS's.

6.4.3 Paper: C3

Title: Creating a documentation system to support the development and maintenance of product configuration systems

Purpose:

This paper describes how the definitions of a documentation system to support the development and maintenance of product configurators have been converted into a software prototype, and presents what has been learned from the evaluation of the prototype.

Prototyping:

Based on the definitions produced in papers C1 and C2, a software prototype was created. The prototype was created with the aim of evaluating definitions related to the modelling environment and not to more administrative functionalities, such as version control, change requests and user access control. This focus was chosen due to the fact that no existing software includes this kind of modelling environment, contrary to the more administrative functionalities, which can be found in numerous software systems. The prototype excluded a few of the elements of the modelling environment according to the ones defined in papers C1 and C2, due to time constraints. The prototype was created by using C# .Net, and approximately 200 hours were spent on design of software architecture and programming. The prototype includes three views: CRC-card, PVM, and class diagram. In figure 25, a screen-dump of the CRC-card view of the prototype is shown.

Research method:

The evaluation of the prototype was made in two ways: by typing data from a PVM model from an ongoing configuration project into the prototype, and by presenting it to knowledge engineers in two companies with much experience with the use of configurators in order to compare the solution to their existing documentation systems. The prototype was presented to one knowledge engineer of the one company, and three knowledge engineers of the other company. The presentations where made by demonstrating the functionality of the prototype and answering questions that arose. After the presentations, unstructured (group) interviews with the knowledge engineers were carried out with the focus on their views of the strengths and limitations of the prototype in general and in comparison to their existing technology. But, although this kind of test gives some indication of the usefulness of the particular software, more is needed to be able to present solid conclusions. To gain real insight into the applicability of the system, different companies would have to test the prototype or even implement it in their daily operations. However, this was not possible in this context, due to time constraints and because of unfinished elements of the prototype.

Results:

The building of the PVM model from the ongoing configuration project in the prototype showed that the prototype could hold most of the information of this model, but that some areas of the software should be improved: there should be better possibilities for stating comments in the CRC-cards; better support of constraints being formulated in tables; and it should be possible to place boxes for comments and constraints next to classes of PVMs and class diagrams. However, this was in some part because all the definitions in papers C1 and C2 had not been included in the prototype.

Both of the interviewed companies stated that a further developed version of the prototype would be very likely to provide significant benefits for them, and that they would be interested in using further developed versions of such software. The main kinds of further developments requested by the companies in order for the prototype to be a real alternative to their existing solutions involve the inclusion of a list of administrative functionality and the possibility of import/export from/to a configurator.

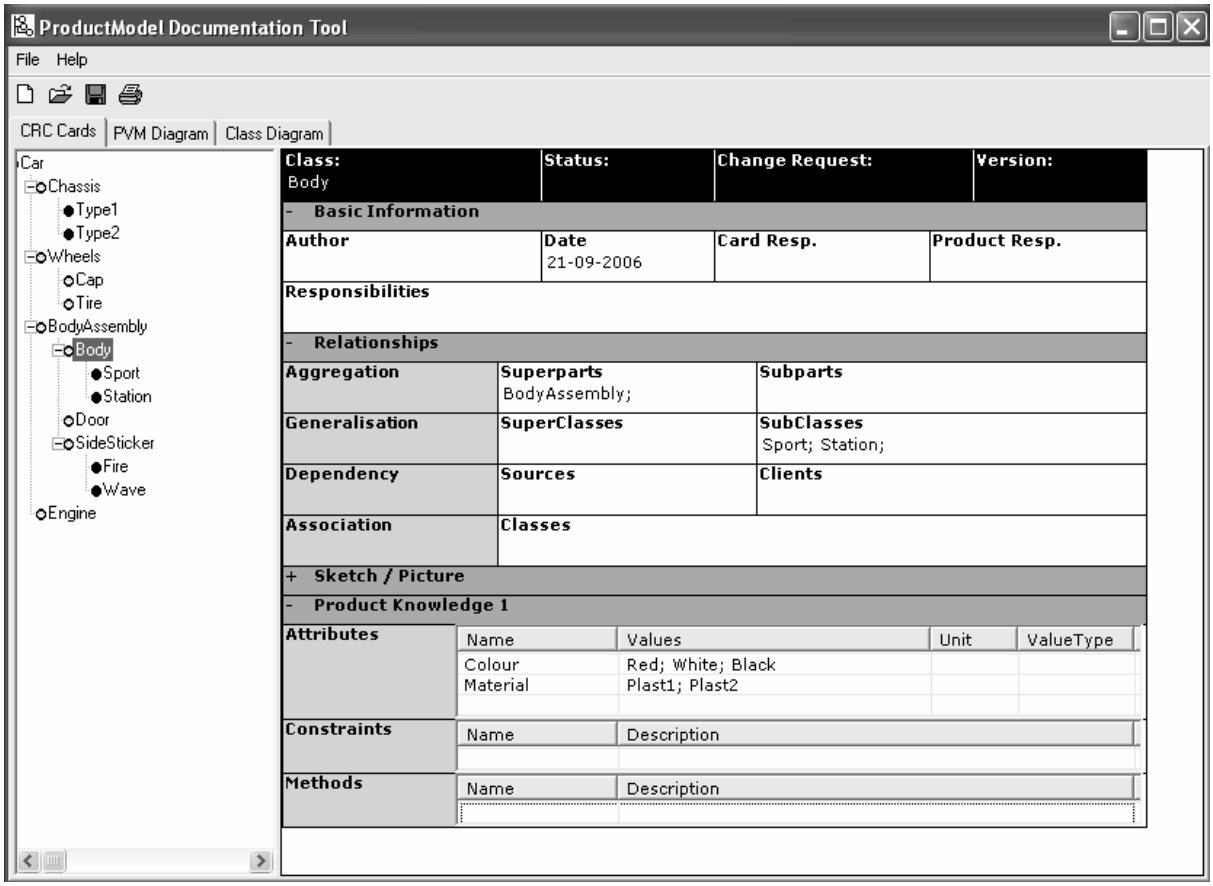


Figure 25: Documentation system prototype

Conclusions:

The paper concludes that the definitions provided in C1 and C2 can be converted into a software application and provide an adequate basis for this development. Furthermore, the paper argues that the creation and evaluation of the prototype have provided an important basis for the development of a complete documentation system. If such a system fulfils expectations, this could have a significant impact on the way in which development and maintenance are carried out in configuration projects, and contribute to higher success rates of such projects.

Chapter 7: Conclusion

7.1 Research questions

In this section, the answers to the seven research questions are presented. Obviously, these answers overlap the descriptions of the papers in chapter 6. However, this section has more specific focus on the research questions, and a broader perspective is applied.

7.1.1 Question 1

1) Does the use of the term 'tacit knowledge' in configuration literature comply with the original meaning of the term, and does it make sense to apply this term in configuration research?

Paper A1 shows that the way the term 'tacit knowledge' is applied in much configuration literature does not comply with the meaning in its philosophical roots. It also shows that a similar 'misuse' of the term is also present in much knowledge management literature, from which it can be assumed that configuration researchers have taken the term. The way the term is used in configuration literature implies that instead of referring to deep and inarticulate knowledge, the term is also used to describe all sorts of articulable knowledge, which is unknown to an observer at some point in time. Thus, the term loses its clarity and does not tell us much about the nature of the knowledge in question. For this reason, it is recommended that, in a product configuration context, the term 'tacit knowledge' only be applied to inarticulate knowledge.

However, this strict definition of 'tacit knowledge' implies that such knowledge does not seem particularly relevant for describing the difficulties in representing domain knowledge in configuration projects. This argument is supported by the studies of four configuration projects described in paper A2, where in two cases, it was claimed that tacit knowledge was 'seldom relevant' and in two other cases 'never relevant'. The knowledge needed to specify a product (e.g. if two specific components may be combined or in which dimensions a particular component is available) should not be confused with the kind of knowledge needed for riding a bicycle or recognizing the face of another person, which are examples of tacit knowledge. This is not to say that tacit knowledge is not involved in the process of specifying products, but that the focus on this is wrong, because it is knowledge at a much less deep level that is interesting in such a context.

Another point is that the commonly applied distinction between tacit and explicit knowledge may not be a particularly useful way of describing knowledge in a configuration context; rather, it presents a great risk of misunderstandings. The fact that some knowledge is tacit may not necessarily represent any difficulty with regard to creating a configurator, just as explicit knowledge (information) cannot by definition be considered to be easy to convert into a model in the knowledge base of a product configurator; for instance, if the knowledge engineer does not understand the retrieved information, or the retrieved information contradicts other retrieved information.

All in all, it is recommended to avoid diluting the term 'tacit knowledge', and to avoid describing knowledge in terms of being tacit and explicit when wanting to say something about knowledge complexity in configuration projects.

7.1.2 Question 2

2) Can the knowledge/information that a domain expert possesses and delivers to a knowledge engineer in a configuration project be categorized in a better way than the tacit-explicit knowledge distinction?

The answer to question 1 gives the reasons why the distinction between tacit and explicit knowledge can be rather unsuitable for telling something about knowledge in a configuration project. Paper A2 proposes a classification of the types of information domain experts deliver and do not deliver to knowledge engineers. Since this classification only covers a small part of the configurator

development process, this can be seen as the beginning of a more complete theory about knowledge and information in configuration projects. The proposed classification of information includes: concealed information, unrecognized information, non-possessed information, incorrect information, irrelevant information, inarticulate information, contradicting information, and relevant explicit information. The first three types can be classified as information that does not leave the domain expert; the next two types comprise information that is not usable; the next two comprise information that requires analysis; and the last comprises directly usable information. As explained in paper A2, it can be discussed whether 'pure explicit knowledge' actually exists, or whether all explicit knowledge is underlain by tacit knowledge. For this reason, the term 'information' is used in the classification instead of 'knowledge'.

The proposed model has been presented to four experienced knowledge engineers for the purpose of falsifying the model by finding types of information outside the model while still within the defined context of focus. The knowledge engineers were also asked to provide justification that all the defined types of information are relevant, by describing situations where they believed they had encountered the eight types of information. None of the knowledge engineers managed to falsify the model, and the existence of all eight types of information was supported by descriptions of concrete situations. While not being seen as conclusive evidence that the proposed model is adequately extensive and does not include irrelevant classes, strong tendencies toward such a conclusion can be claimed.

All in all, it can be concluded that the proposed classification offers a more nuanced and unambiguous basis than the applied tacit-explicit distinction in configuration literature.

7.1.3 Question 3

3) What are the limitations of applying the PVM formalism, and how can the formalism be altered in order to solve such limitations?

By studying existing research data, conducting interviews and observing the use of the PVM technique, five shortcomings of the notation were discovered: 1) a need for extended formalism of notation; 2) a solution of possible inexpediencies when using kind-of structure; 3) possibilities of graphically displaying constraints on relations; 4) preparation of the notation for IT-supported modelling; and 5) utility and instruction in applying notation for modelling the product through life-phase systems. To solve such problems, paper B1 proposes a new diagram type, named Product Family Diagrams (PFDs). Compared to PVMs, PFDs include: various new modelling constructs; a solution of some inexpediency when using kind-of structure (tables to avoid redundancy in shown kind-of dependant variables); a new principle for graphically displaying constraints on relations; a strict and extensive definition of the formalism; and a principle for the modelling of product-life-phase-systems.

Tests of the PFD notation on fictive modelling problems showed that the new notation solves to some extent the problems encountered when applying the PVM technique. However, since the notation was only tested on specific modelling problems, applying the proposed notation to other types of problems may show that altering the existing PVM notation have resulted in new modelling limitations or weaknesses in other contexts. Therefore, in order to arrive at something more conclusive, more experiments would have to be carried out. But the great shortcoming of the knowledge about the applicability of the new notation is that the usability dimension has not been investigated. This does not disqualify the research that has been carried out, but it does imply that more research is needed to say something conclusive about the general usefulness of the proposed notation. In line with this, having reflected over the notation, I suspect that some of the user-friendliness of the PVM notation may be lost when dealing with some types of modelling problems. I would therefore be hesitant to recommend applying the PFD notation in many types of projects. However, in some special contexts, the PFD notation seems very useful, namely in projects with simple product architectures and many variable values that depend on kind-of classes. These factors characterized a modelling task in a project at American Power Conversion, which I discovered had been carried out using the proposed notation (Jacobsen and Kristensen, 2006). The following evaluations of the switch from PVMs to PFDs in the project were made:

"Because of the new method [PFDs instead of PVMs] it was possible to simplify the SVM [Service Variant Master] considerably without leaving out information about variants..." and
"The result [of applying PFDs instead of PVMs] is, therefore, a very compact, but still easy accessible and operational SVM, where all necessary information can be found quickly" (trans. from Jacobsen and Kristensen, 2006)

To summarize, some shortcomings have been pointed out in the PVM technique when applied in practice, and a new notation, named PFDs, has been produced to solve some of these problems. Based on the current evidence, its use can only be justified for modelling problems with certain characteristics. Therefore, in order to deal with more common modelling problems in configuration projects while avoiding some of the disadvantages of the existing PVM formalism, I recommend applying the PSCD definition in paper B3 or the PVM definition in paper C1.

7.1.4 Question 4

4) What are the actual differences between using PVMs and class diagrams for modelling problems in configuration projects?

The differences between PVMs and class diagrams can be organized into expressional strength and usability issues. In papers B2 and B3, the advantages of class diagrams compared to PVMs were found to be:[38]

1) A more unambiguous and well-defined notation formalism
2) Widespread use within software development (better known by e.g. software developers)
3) The possibility of modelling other aspects than structure with the same language (i.e. UML)
4) Much standard software supports the elaboration of class diagrams, while the PVM notation is not supported by any standard software
5) More predefined relationship types than in the PVM notation
6) A means of creating new types of model elements within normative use of the notation (i.e. stereotypes)
7) A larger range of predefined model elements to symbolize various concepts
8) Notation for modelling relations between different models (i.e. package diagrams)
9) Allows the display of classes which belong to multiple wholes, or inherits from more than one class without having to show the same class more than once in the diagram
10) A more flexible paper format of models, since model elements can be placed according to any preferred height-width ratio, while PVMs tend to become long and slim

The PVM technique, on the other hand, only seems to have usability-related advantages over class diagrams, i.e. the learnability of the notation (how fast users can learn and start using the technique) plus the advantage that it is well-suited for stepwise revision of its content. In contrast to the advantages of class diagrams, the learnability and review-related advantages of PVMs are not facts that can be pointed out, but issues dependent on context. To support these claims, case studies, observations, experiments and explanations have been conducted. Interviews with persons from four companies showed that they were all of the opinion that PVMs are easier to learn than class diagrams. Two of the companies even had abandoned the use of class diagrams for PVMs for certain tasks. However, several of the interviewed persons stated that they had faced problems due to the expressional limitations of the technique.

The observations showed that the PVM notation was understood by some domain experts with minimal modelling experience and hardly any introduction. However, since similar observations of the use of class diagrams have not been conducted, comparisons are not possible. The experiments that investigated the usability of PVMs compared to class diagrams did not show adequate variance in dependence variables, which when also recognizing the small sample size and significant uncertainty

[38] It should be mentioned, that the PVM formalism to some extent could be extended to include points 5, 6, 7 and 8. But this is as a basis point outside the defined PVM formalism (Hvam et al., 2007a).

factors, supported in a convincing way the claim that PVMs are more user-friendly. However, when comparing the produced models from the experiments, it seems fair to claim that the PVMs produced were more strictly organized and therefore easier to overview (shown in appendix 4). This phenomenon can be explained by the fact that PVMs, compared to class diagrams, include placement rules that govern how users place model elements. This includes the rule that different relationship types in PVMs are placed in separate columns, which obviously affects how easy it is to distinguish between relationship types. Furthermore, the placement rules of PVMs produce a reading pattern that is closer to text than normally drawn class diagrams. This characteristic makes PVMs particularly well-suited for pointwise reviews in many cases. Finally, the way of showing aggregation in PVMs resembles a well-known 'symbol for this', which is found for instance in the way BOMs and tables of contents are structured, which in many cases would make the meaning intuitively known, compared to the diamond symbol from class diagrams.

7.1.5 Question 5

5) How can the migration of information from PVMs to class diagrams be avoided while not losing the benefits of the application of both techniques?

By applying the placement rules of PVMs to class diagrams, a new layout principle called Product Structure Class Diagrams (PSCDs) is proposed in paper B3. Under the assumption that this diagram holds both the expressional strengths of class diagrams and the usability of PVMs, there are few arguments against using PSCDs instead of PVMs followed by class diagrams in configuration projects.

It has been shown that the first nine of the ten advantages of class diagrams that were pointed out in the answer to question 4 can to a large extent be maintained, in spite of applying the placement rules of PVMs to class diagrams. The usability of PSCDs was compared to class diagrams and PVMs in an experiment, but due to small sample size and significant uncertainties concerning some of the control variables, together with the limited variance of dependent variables, the experiment did not allow solid conclusions to be drawn concerning errors and how easy it is to create models. However, comparisons of the models drawn in the experiment strongly indicate that the use of PSCDs will result in models that are easier to overview and review than class diagrams, and comparable to PVMs in this respect (see appendix 4). However, since experiences from practice concerning the use of PSCDs have not yet been acquired, other unforeseen problems from the use of PSCDs could emerge. But based on the evidence produced so far, PSCDs seem in many cases to be a good alternative to the use of PVMs, especially in approaches where PVM models are to be migrated into class diagrams.

7.1.6 Question 6

6) How can models with overlapping information in configuration projects be maintained, while avoiding having to update the same information in several places?

To allow maintenance of models with overlapping information, paper B4 proposes a modelling principle together with two possible software solutions. The basic idea of the modelling principle is that instead of managing models with overlapping information separately, they can be maintained in a common model with different views. This implies that much redundant work of updating information and ensuring consistency across models can be avoided. Through the use of stereotypes, the modelling principle includes definitions of constructs for tagging model elements[39] according to which view they belong to. Furthermore, the proposed modelling principle includes definitions of how to handle model elements in cases where classes are represented as attributes in other model views. Also included in the principle is a definition of how to handle specialization classes, which in another model view should be included in their generalization class. However, the proposed modelling principle presumes a software solution in order to be applied in practice.

[39] Classes, attributes, methods, constraints and relationships.

Two software support solutions have been proposed: MS Visio (by turning layers on/off) and inclusion of the modelling principle in a configuration documentation system. The main weakness of the Visio solution is that model elements that are turned off in particular views leave blank spaces in the model, because the solution does not support the use of 'intelligent' placement rules. Also, the same notation has to be used in different models. Integrating the modelling principle into a configuration documentation system could eliminate such problems, whereas the MS Visio alternative is most useful in cases with little individual content for each model view.

The modelling principle has been tested by applying it to a fictive modelling problem with overlapping analysis and design models. It was shown that the defined constructs for managing elements according to views allow two models with overlapping content to be merged. Furthermore, parts of the modelling principle have been applied in an actual configuration project, using the MS Visio solution. The result of this study was that in the specific case, the use of MS Visio provided adequate software support and eliminated the need for having separate models with overlapping content. In spite of the blank spaces in the models presented to the domain experts, their reaction was not negative. However, in the investigated case, only few model elements belonged to the individual views.

All in all, it has been shown that the principle proposed for managing models with overlapping content can be very useful in configuration projects. The major problem in relation to the application of the principle in practice is that it currently is only possible in MS Visio or similar programs with all the limitations such include. The development of a configuration documentation system could increase the usefulness of the proposed modelling principle significantly.

7.1.7 Question 7

7) What are the necessary definitions for the creation of a documentation system that supports the CPM-procedure?

Paper C1 argues that for creating a configuration documentation system to support the modelling techniques of the CPM-procedure, the needed specifications must at least include: a formalized PVM notation that allows software support; a definition of the subset of class diagrams to be included; a definition of which package diagram elements to include; an extended CRC-card definition; definitions of how to handle relationships between PVMs, class diagrams and CRC-cards; and the basic windows of the software system. Paper C1 defines all these elements, except a full CRC-card layout. The definitions of how to handle relationships between PVMs, class diagrams and CRC-cards are based on the use of stereotypes for handling: common components (reusable global models); classes that share CRC-cards; mapping from CRC-card navigation tree to class diagrams (to express multiple inheritance in a three); and constraint classes.

Since investigations of two of the companies in Denmark with the most structured procedures for documenting configurator knowledge showed that their layout differs very much from the current CPM definitions, and also from each other, paper C2 concludes that a generic and extended CRC-card definition is needed in order to support the needs of different companies. Thus, paper C2 defines a much extended CRC-card layout, where relevant fields are to be chosen and new fields created, depending on the individual configuration project.

It has been shown that at least the central elements of the definitions of paper C1 and C2 can be transformed into a software system, as described in paper C3. The prototype has been evaluated by putting product data from an ongoing configuration project into the prototype and by presenting it to two companies. Based on the current tests and feedback on the prototype, the design of a documentation system seems to match the needs of companies in configuration projects. However, more insight into how such a system functions in a real configuration project is needed. To start with, this requires further software development than the mentioned prototype.

All in all, an important basis for the creation of a documentation system that supports the CPM-procedure has been established. It should also be noted that in spite of some years of research on such a system at CPM, the created prototype is in fact the first system ever that supports PVMs, class diagrams and CRC-cards in an integrated fashion. This is therefore a big step toward fulfilling the ambition of CPM that such a documentation system will some day emerge.

7.2 Methodological reflections

As mentioned, my research has been carried out from a critical realist perspective in relation to views on ontology, epistemology and methodology. But, as mentioned, I have not included more specific critical realistic terminology or explanation models in my papers, mainly because of their focuses and intended audience. On the other hand, the extended description of the methodology of the papers in chapter 6 allowed me to clarify my methodological approach, which makes the critical realist dimension more obvious. Also, it must be recognized that a focus on understanding reality has not been the central element in most of my research, compared to proposing new solutions to practical problems. In my opinion, the latter alone is not science, but identifying problems and testing solutions belong to the scientific sphere, and the approaches applied here are not in contradiction with critical realism.

All in all, I therefore believe that the approach I have applied in answering the seven research questions fits well into the frame of critical realism. Some of the points that can be mentioned are:

- Critical realism's belief that all knowledge is fallible while independent reality does exist shines through the way conclusions are formulated in my research. In recognition that it is not possible to establish absolute truths, the focus has been on finding conditional tendencies towards truths, i.e. 'good reasons to believe'.
- The great variety of applied research methods (different interview techniques, observation, case studies, action research, experiments and prototyping) underlines my subscription to the point of critical realism that maintains that the research problem should define the research method and not the other way around.
- The interdisciplinary approach that is advocated by critical realism is to some extent found in the research in that it includes theory from areas such as operation management, information science, artificial intelligence, engineering design theory, knowledge management and philosophy. However, a quest for explanations at even deeper levels of reality, such as biology or more fundamental physiology, has not been made.
- In answering the research questions, approaches advocated by critical realism were applied, i.e. abstraction, a complementary induction-deduction approach, and retroduction.
- The mission of ensuring clarity of concepts, the importance of which is emphasized by Bhaskar in relation to research on social factors, is reflected in parts of my research. I am of the opinion that it cannot be emphasized enough that, for research that does not build on mathematical expressions, it is essential that applied terms and concepts are unambiguous in order to for such research to form a foundation for further research.

By recognizing these points regarding my scientific approach, I position myself with a clear distance to positivist notions (including critical rationalism). In particular, I do not formulate a hypothesis for verification or falsification in the traditional sense, while I believe it is possible to learn much from this approach when dealing with social factors. Rather my focus has often been more on investigating which factors are relevant in order for a certain phenomenon to occur, i.e. something that resembles a retroduction approach. On other hand, I also distance myself in my research from a traditional social constructivist position, since I do not focus much on explaining how some knowledge or technology has been formed by social factors. Instead, my research indirectly reflects the basic assumption that it is an independent natural and social reality which is being investigated.

7.3 Summary of contributions

In addition to the answers to the research questions, the most significant contributions of this PhD thesis can be summarized brief as follows:

1) An overview of relevant research carried out at four of the most important groups involved in configuration research: University of Klagenfurt, Helsinki University of Technology, Forza and Salvador, and the Technical University of Denmark (chapter 2).

2) A clarification of the meaning of some important terms and concepts that are commonly applied in configuration research (chapter 4). Hereunder, there is a discussion of the problems of labelling ETO-companies as mass customizers (Haug et al., 2007, found in Appendix 1).
3) A possible solution to the conflict between scientific ideals (critical rationalism) and the way product configuration research is actually carried out at my department. The thesis proposes a shift from critical rationalism to critical realism, which implies maintaining the ideals of realism and fallibilism, but recognizing that unlimited and undistorted access to reality is not possible (chapter 5).
4) Clarification of the meaning of the concept of 'tacit knowledge' and description of the unfortunate consequences of the current use (misuse) in much configuration literature (paper A1).
5) A classification of the information possessed by a domain expert in a knowledge acquisition situation. This classification may be useful, both for understanding the acquisition process in practice and to build on in future configuration research (paper A2)
6) A new notation technique, named PFDs (Product Family Diagrams). Based on application in practice, this seems to be useful when modelling product families with many attribute values that are dependent on kind-of classes (paper B1).
7) Insights into the differences between using PVMs and class diagrams to create conceptual models in configuration projects by analysing the notations defining their individual strengths and weaknesses in comparison with each other (paper B2).
8) A layout principle named PSCDs (Product Structured Class Diagrams) for class diagrams, which integrates the placement rules of PVMs into the class diagram notation. Investigations have been made, which indicate that the major advantages of both diagrams have been achieved through this merger (paper B3).
9) A modelling principle that allows separate models with overlapping information to be maintained in a common model, hereby avoiding redundant modelling work and problems in ensuring consistency across models. In addition, two technical solutions for software support of the modelling solution principle have been defined, and one of these has been applied in practice with good results (paper B4).
10) A definition of a documentation system that supports the modelling techniques of the CPM-procedure, where it has been shown that these definitions can be converted into a software system (paper C1; C2; C3).
11) A new generic CRC-card definition that allows companies to adapt the extensive basic layout (compared to existing CPM definitions) to individual needs (paper C2).
12) The creation of the first software prototype that supports the modelling techniques of the CPM-procedure in an integrated fashion (paper C3).

7.4 Future research

Overall, this thesis has dealt with the three topics: domain expert knowledge, knowledge representation techniques, and documentation of configuration knowledge. Although having produced significant contributions to these areas, much research remains in all three areas.

Concerning the understanding of knowledge and information involved in the process of developing and maintaining product configurators, this work has only just begun by analysing 'the information that domain experts do and do not deliver to knowledge engineers'. Many other such processes need to be analysed and described in order to create a more complete framework that describes the information and knowledge involved in configuration projects.

Concerning graphical representation techniques and modelling principles, several of these have emerged during this PhD project. Most of these propositions need to be further investigated, e.g. by going deeper into theory of usability and semiotics, together with conducting more usability experiments and retrieving experience from application in practice. Also, many ends need to be tied together for a better overview of which approaches to choose in a particular project.

Concerning the ambition of CPM to have a documentation system that supports the modelling techniques of the CPM-procedure, what can be seen as a major breakthrough was made during this

PhD project, in that the first prototype for such a system emerged from the research produced. However, there is still some work to be done before such a system can be applied in industry, which is however mostly of a software development character as opposed to software specification.

All in all, this PhD has filled several holes in the configuration literature concerning domain expert knowledge, knowledge representation and documentation systems. But by doing this, new areas have emerged that need investigation. This is a task that I hope to be granted the privilege of carrying out in the nearest future.

References

Aamodt and Plaza, 1994	Aamodt, A. and Plaza, E. (1994): "Case-based reasoning: Foundational issues, methodological variations, and system approaches", in AI Communications, 7(1): 39-59.
Aldanondo et al., 2000:	Aldanondo, M., Rouge, S. and Veron, M. (2000): "Expert configurator for concurrent engineering: Cameleon software and model", in Journal of Intelligent Manufacturing, 11(2): 127-134.
Andreasen, 1992	Andreasen, M.M. (1992): "Designing on a 'Designer's Workbench' (DWB)", in Proceedings of the 9th WDK Workshop, Rigi, Switzerland.
Andreasen, 1998	Andreasen, M.M. (1998): "Conceptual design capture", in Proceedings of EDC ’98 - Design Reuse, Brunel University.
Archer, 1995	Archer, M. (1995): "Realist social theory - The morphogenetic approach", Cambridge: Cambridge University Press.
Archer et al., 1998	Archer, M., Bhaskar, R., Collier, A., Lawson and Allan, N. (eds.) (1998): "Critical realism: Essential readings", London: Routledge.
Ardissono et al., 2003	Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Petrone, G., Schäfer, R. and Zanker, M. (2003): "A framework for the development of personalized, distributed web-based configuration systems", in AI Magazine, 24(3): 93-108.
Asikainen et al., 2004	Asikainen, T., Soininen, T. and Männistö, T. (2004): "A Koala-based approach for modelling and deploying configurable software product families", in Lecture Notes in Computer Science, 3014: 225-249.
Asikainen et al., 2007	Asikainen, T., Männistö, T. and Soininen, T. (2007): "Kumbang: A domain ontology for modelling variability in software product families", in Advanced Engineering Informatics, 21(1): 23-40.
Barker and O'Connor, 1989	Barker, V.E., O'Connor, D.E., Bachant, J. and Soloway, E. (1989): "Expert systems for configuration at Digital: XCON and beyond", in Communications of the ACM, 32(3): 298-318.
Beck and Cunningham, 1989	Beck, K. and Cunningham, W.A. (1989): "A laboratory for teaching object-oriented thinking", in Proceedings of OOPSLA '89 and Special issue of SIGPLAN Notices, 24(10): 1-6.
Berger and Luckmann, 1987	Berger, P.L. and Luckmann, T. (1987): "Den samfundsskabte virkelighed" (orig. The social construction of reality - A treatise in the sociology of knowledge, 1966), Copenhagen: Lindhardt and Ringhof.
Bertalanffy, 1972	Bertalanffy, L.V. (1972): "The history and status of general systems theory", in Academy of Management Journal, 15(4): 407-426.
Birmingham et al., 1988	Birmingham, W.P., Brennan, A., Gupta, A.P. and Siewiorek, D.P. (1988): "MICON - A single board computer synthesis tool", in IEEE Circuits and Devices Magazine, 4(1): 37-46.
Brachman and Levesque, 2004	Brachman, R. J. and Levesque, H. J. (2004): "Knowledge representation and reasoning", New York: Morgan Kaufmann Publishers.
Brier, 2006	Brier, S. (2006): "Informationsvidenskabsteori", 2nd edition, Frederiksberg: Forlaget Samfundslitteratur.
Buch-Hansen and Nielsen, 2007	Buch-Hansen, H. and Nielsen, P. (2007): "Kritisk realisme", Frederiksberg: Roskilde Universitetsforlag.
Callon, 1987	Callon, M. (1987): "Society in the making: The study of technology as a tool for sociological analysis", in W.E. Bijker, T.P. Hughes and T.J. Pinch (Eds.), The Social Construction of Technical Systems: New Directions in

the Sociology and History of Technology (pp. 83-103), London: MIT Press.

Cawsey, 1998 — Cawsey, A. (1998): "The essence of artificial Intelligence", Harlow: Prentice Hall.

Chalmers, 1999 — Chalmers, A.F. (1999): "What is this thing called Science?", 3rd edition, Birkshire: Open University Press.

Chao and Chen, 2001 — Chao, P.Y., Chen, T.T. (2001): "Analysis of assembly through product configuration", in Computers in Industry, 44(2): 189-203.

Collin, 2003 — Collin, F. (2003): "Konstruktivisme", Frederiksberg: Roskilde Universitetsforlag.

Cunis et al., 1989 — Cunis, R., Günter, A., Syska, I., Peters, H. and Bode, H. (1989): "PLAKON - An approach to domain-independent construction", in Proceedings of the 2nd International Conference on Industrial and Engineering Applications of AI and Expert Systems (pp. 866–74), Tullahoma, TN.

Danermark et al., 2002 — Danermark, B., Ekström, M., Jakobsen, L. and Karlsson, J.C. (2002): "Explaining society - Critical realism in the social sciences", London: Routledge.

Edwards and Pedersen, 2004 — Edwards, K. and Pedersen, J.L. (2004): "Product configuration systems - Implications for product innovation and development", in Proceedings of International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems (PETO), Lyngby, Denmark.

Edwards and Riis, 2004 — Edwards, K. and Riis, J. (2004): "Expected and Realized Costs and Benefits when Implementing Product Configuration Systems", in Proceedings of the 8th International Design Conference (DESIGN 2004), Dubrovnik, Croatia.

Edwards et al., 2005 — Edwards, K., Hvam, L., Pedersen, J.L., Møldrup, M. and Møller, N. (2005): "Udvikling og implementering af konfigureringssystemer: Økonomi, Teknologi og Organisation", Final report from PETO research project, Department of Manufacturing Engineering and Management, Technical University of Denmark.

Felfernig et al., 2000a — Felfernig, A., Friedrich, G.E. and Jannach, D. (2000a): "UML as domain specific language for the construction of knowledge-based configuration systems", in International Journal of Software Engineering and Knowledge Engineering, 10(4): 449-469.

Felfernig et al., 2000b — Felfernig, A., Friedrich, G., Jannach, D and Zanker, M. (2000b): "A framework for the development of cooperative configuration agents", in Lecture Notes in Artificial Intelligence, 1821: 24-33.

Felfernig et al., 2000c — Felfernig, A., Friedrich, G., Jannach, D. and Zanker, M. (2000c): "Integrating knowledge-based configuration systems by sharing functional architectures", in Lecture Notes in Artificial Intelligence, 1937: 312-327.

Felfernig et al., 2000d — Felfernig, A., Jannach, D. and Zanker, M. (2000d): "Contextual diagrams as structuring mechanisms for designing configuration knowledge bases in UML", in Lecture Notes in Computer Science, 1939: 240-254.

Felfernig et al., 2001 — Felfernig, A., Friedrich, G. and Jannach, D. (2001): "Conceptual modeling for configuration of mass-customizable products", in Artificial Intelligence in Engineering, 15(2): 165-176.

Felfernig et al., 2002 — Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M. and Zanker, M. (2002): "Acquiring configuration knowledge bases in the semantic web using UML", in Lecture Notes in Artificial Intelligence, 2473: 352-357.

Felfernig et al., 2003 — Felfernig, A., Friedrich, G., Jannach, D., Stumptner, M. and Zanker, M. (2003): "Configuration knowledge representations for semantic web

applications", in Artificial Intelligence for Engineering Design Analysis and Manufacturing, 17(1): 31-50.

Felfernig et al., 2004 — Felfernig, A., Friedrich, G., Jannach, D. and Stumptner, M. (2004): "Consistency-based diagnosis of configuration knowledge bases", in Artificial Intelligence, 152(2): 213-234.

Fleischanderl et al., 1998 — Fleischanderl, G., Friedrich, G., Haselböck, A., Schreiner, H. and Stumptner, M. (1998): "Configuring large systems using generative constraint satisfaction", in IEEE Intelligent Systems, 13(4): 59-68.

Flyvbjerg, 2001 — Flyvbjerg, B. (2001): "Making social science matter", Cambridge: University Press.

Forza and Salvador, 2002a — Forza, C. and Salvador, F. (2002a): "Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems", in International Journal of Production Economics, 76(1): 87-98.

Forza and Salvador, 2002b — Forza, C. and Salvador, F. (2002b): "Product Configuration and inter-firm co-ordination: An innovative solution from a small manufacturing enterprise", in Computers in Industry, 49(1): 37-46.

Forza and Salvador, 2007 — Forza, C. and Salvador, F. (2007): "Product information management for mass customization", Hampshire: Palgrave Macmillan.

Forza et al., 2006 — Forza, C., Trentin, A. and Salvador, F. (2006): "Supporting product configuration and form postponement by grouping components into kits: The case of MarelliMotori", in International Journal of Mass Customization, 1(4): 427-444.

Fowler, 2005 — Fowler, M. (2005): "UML distilled: A brief guide to the standard object modeling language", 3rd edition, Boston, MA: Addison-Wesley.

Fuglsang, 2005 — Fuglsang, F. (2005): "Systemteori og funktionalisme", in Videnskabsteori i samfundsvidenskaberne (pp. 115-144), Frederiksberg: Roskilde Universitetsforlag.

Fuglsang and Olsen, 2005 — Fuglsang, F. and Olsen, P.B. (2005): "Introduktion", in Videnskabsteori i samfundsvidenskaberne (pp. 7-51), Frederiksberg: Roskilde Universitetsforlag.

Giarratano and Riley, 2005 — Giarratano, J.C. and Riley, G.D. (2005): "Expert systems - Principles and programming", 4th edition, Boston, MA: Thomson Course Technology.

Gordon, 2000 — Gordon, J.L. (2000): "Creating knowledge maps by exploiting dependent relationships", in Knowledge Based Systems", 13(2-3): 71-79.

Guéhéneuc and Albin-Amiot, 2004 — Guéhéneuc, Y.-G., and Albin-Amiot, H. (2004): "Recovering Binary Class Relationships: Putting Icing on the UML Cake", in Proceedings of OOPSLA'04 (pp. 301-314), Vancouver, Canada.

Hansen, 2003 — Hansen, B.L. (2003): "Development of Industrial Variant Specification Systems", PhD dissertation, Department of Manufacturing Engineering and Management, Technical University of Denmark.

Hansen and Hvam, 2002 — Hansen, B.L. and Hvam, L. (2002): “Experiences with a procedure for modelling product knowledge and building product configurators - at an American manufacturer of air conditioning equipment”, in Proceedings of the 15th European Conference on Artificial Intelligence, Lyon, France.

Hansen et al., 2003 — Hansen, B.L., Riis, J.R. and Hvam, L. (2003): "Specification process reengineering: concepts and experiences from Danish industry", in Proceedings of the 10th ISPE International Conference on Concurrent Engineering, Madeira, Portugal.

Harlou, 2006 — Harlou, U. (2006): "Developing product families based on architectures – Contribution to a theory of product families", PhD dissertation,

Department of Mechanical Engineering, Technical University of Denmark.

Hartnack, 1979 — Hartnack, J. (1979): "Fra Kant til Hegel. En nytolkning", Copenhagen: Berlingske Forlag.

Haug et al., 2007 — Haug, A., Ladeby, K. and Edwards, K. (2007): "Reflections on the transition from ETO to Mass Customization", in Proceedings of MCPC-2007, Boston, MA.

Haugeneder et al., 1985 — Haugeneder, H., Lehmann, E. and Struss, P. (1985): "Knowledge-based configuration of operating systems - problem in modeling the domain knowledge", in Proceedings of the GI Congress on Knowledge-Based Systems (pp. 121-134), Munich, Germany.

Heinrich and Jüngst, 1991 — Heinrich, M., and Jüngst, E. (1991): "A resource-based paradigm for the configuring of technical systems from modular components", in Proceedings of the 7th IEEE Conference on Artificial Intelligence Applications (pp. 257–264), Miami Beach, FL.

Hopgood, 2000 — Hopgood, A.A. (2000): "Intelligent systems for engineers and scientists", 2nd edition, London: CRC Press.

Hvam, 1994 — Hvam, L. (1994): "Anvendelse af Produktmodellering - set ud fra en arbejdsforberedelsessynsvinkel", PhD dissertation, Department of Industrial Management and Engineering, Technical University of Denmark.

Hvam, 1996 — Hvam, L. (1996): "Application of product modelling – seen from a work preparation viewpoint" (English translation of Hvam, 1994), Department of Industrial Management and Engineering, Technical University of Denmark.

Hvam, 1998 — Hvam, L. (1998): "The rulers factory - a tool for learning product modeling techniques", in Computers & Industrial Engineering, 35(1-2): 29-32.

Hvam, 1999 — Hvam, L. (1999): "A procedure for building product models", in Robotics and Computer-integrated Manufacturing, 15(1): 77-87.

Hvam, 2001 — Hvam, L. (2001): "A procedure for the application of product modelling", in International Journal of Production Research, 39(5): 873-885.

Hvam, 2004 — Hvam, L. (2004): "A multi-perspective approach for the design of Product Configuration Systems – an evaluation of industry applications", in Proceedings of the International Conference of Economic, Technical and Organizational aspects of Product Configuration Systems (PETO), Lyngby, Denmark.

Hvam, 2006a — Hvam, L. (2006a): "Mass customization and configuration of process plants", in International Journal of Mass Customization, 1(4): 445-462.

Hvam, 2006b — Hvam, L. (2006b): "Mass customization in the electronics industry - based on modular products and product configuration", in International Journal of Mass Customization, 1(4): 410-426.

Hvam and Hansen, 1999 — Hvam, L. and Hansen, B. (1999): "Strategic guidelines for application of product models", in Proceedings of the 4th Annual International Conference on Industrial Engineering Theory, Applications and Practice, San Antonio, Texas.

Hvam and Have, 1998 — Hvam, L. and Have, U. (1998): "Re-engineering the Specification Process", in Business Process Management Journal, 4(1): 25-43.

Hvam and Malis, 2001 — Hvam, L. and Malis, M. (2001): "A knowledge based documentation tool for configuration projects", in Proceedings of World Congress on Mass Customization and Personalization, Hong Kong.

Hvam and Riis, 2003	Hvam, L., Riis, J. and Hansen, B.L. (2003): "CRC cards for product modelling", in Computers in Industry, 50(1): 57-70.
Hvam et al., 2002	Hvam, L., Riis, J. and Malis, M. (2002): "A multi-perspective approach for the design of configuration systems", in Proceedings of the 15th European Conference on Artificial Intelligence, Lyon, France.
Hvam et al., 2004a	Hvam, L., Malis, M., Hansen, B. and Riis, J. (2004): "Reengineering of the quotation process - Application of knowledge based systems", in Business Process Management Journal, 10(2): 200-213.
Hvam et al., 2004b	Hvam, L., Mortensen, N.H. and Riis, J. (2004b): "Produktkonfigurering" (internal publication used for teaching product configuration), Department of Manufacturing Engineering and management at the Technical University of Denmark.
Hvam et al., 2005	Hvam L., Pape S., Jensen K.L., and Riis, J. (2005): "Development and maintenance of product configuration systems: Requirements for a documentation tool", in International Journal of Industrial Engineering-Theory Applications and Practice, 12(1): 79-88.
Hvam et al., 2006	Hvam L., Pape S. and Nielsen M.K. (2006): "Improving the quotation process with product configuration", in Computers in Industry, 57(7): 607-621.
Hvam et al., 2007a	Hvam, L., Mortensen, N.H. and Riis, J. (2007a): "Produktkonfigurering", Copenhagen: Nyt Teknisk Forlag.
Hvam et al., 2007b	Hvam, L., Mortensen, N.H. and Riis, J. (2007b): "Product Customization" (preliminary English version of Hvam et al., 2007a), to be published by Springer in 2008.
Jackson, 1999	Jackson, P. (1999): "Introduction to expert systems", 3rd edition, Essex: Addison Wesley Longman Limited.
Jacobsen and Kristensen, 2006	Jacobsen, R.S. and Kristensen, S. (2006): "Configuration of services at APC", Masters thesis, Department of Manufacturing Engineering and Management, Technical University of Denmark.
Jespersen, 2005	Jespersen, J. (2005): "Kritisk realisme", in Videnskabsteori i samfunds-videnskaberne (pp. 145-177), Frederiksberg: Roskilde Universitetsforlag.
Jinsong et al., 2005	Jinsong, Z., Qifu, W., Li, W. and Yifang, Z. (2005): "Configuration-oriented product modelling and knowledge management for made-to-order manufacturing enterprises", in International Journal of Advanced Manufacturing Technology, 25(1-2): 41–52.
Johnson and Cassel, 2001	Johnson, P. and Cassel, C. (2001): "Epistemology and work psychology: New agendas", in Journal of Occupational and Organizational Psychology, 74(2): 125-143.
Jónsdóttir, 1998	Jónsdóttir, S.M. (1998): "IT based product models for development of seafood products", PhD dissertation, Department of Industrial Management and Engineering, Technical University of Denmark.
Kneer and Nassehi, 2004	Kneer, G. and Nassehi, A. (2004): "Niklas Luhmann - Introduktion til teorien om sociale systemer", Danish trans. by N. Mortensen, Copenhagen: Hans Reitzels Forlag.
Koch, 2005	Koch, C.A. (2005): "Kritisk rationalisme", in Videnskabsteori i Samfunds-videnskaberne (pp. 79-111), Frederiksberg: Roskilde Universitetsforlag.
Kojo et al., 2003	Kojo, T., Mannisto, T. and Soininen, T. (2003): "Towards intelligent support for managing evolution of configurable software product families", in Lecture Notes in Computer Science, 2649: 86-101.
Kopisch and Günter, 1992	Kopisch, M. and Günter, A. (1992): "Configuration of a passenger aircraft cabin based on conceptual hierarchy, constraints, and flexible control", in Lecture Notes in Artificial Intelligence, 604: 421-430.

Larman, 2002	Larman C. (2002): "Applying UML and patterns", 2nd edition, New Jersey: Prentice Hall.
Latour, 1987	Latour, B. (1987): "Science in action", Cambridge, MA: Harvard University Press.
Law, 1991	Law, J. (Ed.) (1991): "A sociology of monsters: Essays on power, technology and domination", London: Routledge.
Lawson, 1997	Lawson, T. (1997): "Economics & Reality", London: Routledge.
Lawson, 2003	Lawson, T. (2003): "Reorienting economics", London: Routledge.
Lewin, 1946	Lewin, K. (1946): "Action research and minority problems", in Journal of Social Issues, 2: 34-46.
Luhmann, 2003	Luhmann, N. (2003): "Iagttagelse og Paradoks", Danish trans. by H.C. Fink, J. Katlev and O. Thyssen, Copenhagen: Gyldendal.
Lübcke at al., 2001	Lübcke, P. (Eds.) (2001): "Politikens filosofi leksikon", Copenhagen: Politikens Forlag.
Magro and Torasso, 2003	Magro, D. and Torasso, P. (2003): "Decomposition strategies for configuration problems", in Artificial Intelligence for Engineering Design Analysis and Manufacturing, 17(1): 51-73.
Malis, 2005	Malis, M. (2005): "Application of product models in extended enterprises", PhD dissertation, Department of Manufacturing Engineering and Management, Technical University of Denmark.
Männistö et al., 2001	Männistö, T., Peltonen, H., Soininen, T. and Sulonen, R. (2001): "Multiple abstraction levels in modelling product structures", in Data & Knowledge Engineering, 36(1): 55-78.
Marcus et al., 1988	Marcus, S., Stout, J. and McDermott, J. (1988): "VT: an expert elevator designer that uses knowledge-based backtracking", in AI Magazine 9(2): 95-111.
McDermott, 1982	McDermott, J. (1982): "R1: A rule-based configurer of computer systems", in Artificial Intelligence, 19(1): 39-88.
Mittal and Falkenheiner, 1990	Mittal, S. and Falkenheiner, B. (1990): "Dynamic constraint satisfaction problems", in Proceedings of AAAI-90 the Eighth National Conference on Artificial Intelligence (pp. 25-32), Boston, MA.
Mittal and Frayman, 1989	Mittal, S. and Frayman, F. (1989): "Towards a generic model of configuration tasks", in Proceedings of the 11th International Joint Conference on AI (pp. 1395–1401), Detroit, MI.
Mortensen, 1999	Mortensen, N.H. (1999): "Design modelling in a designer's workbench - Contribution to a design language", PhD dissertation, Department of Control and Engineering Design, Technical University of Denmark.
Mortensen, 2004;	Mortensen, N. (2004): "Forord", in Kneer and Nassehi, Niklas Luhmann - Introduktion til teorien om sociale systemer (pp. 7-11), Danish trans. by N. Mortensen, Copenhagen: Hans Reitzels Forlag.
Mortensen et al., 2000	Mortensen, N.H., Yu, B., Skovgaard, H. and Harlou, U. (2000): "Conceptual modeling of product families in configuration projects", in Proceedings at the Workshop at the 14th European Conference on Artificial Intelligence, Berlin, Germany.
Møldrup and Møller, 2004	Møldrup, M. and Møller, N. (2004): "Development and implementation of product configuration systems – a change management perspective", in Proceedings of International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems (PETO), Lyngby, Denmark.

Najman and Stein, 1992	Najman, O. and Stein, B. (1992): "A theoretical framework for configurations", in Proceedings of Industrial and Engineering Applications of Artificial Intelligence and Expert Systems: 5th International Conference (pp. 441–50), Paderborn, Germany.
OMG, 2005	OMG (2005): "Unified Modeling Language: Superstructure: Version 2.0", formal/05-07-04, Nedham, MA.
Pedersen and Edwards, 2004	Pedersen, J.L. and Edwards, K. (2004): "Product configuration systems and productivity", in Proceedings of International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems (PETO), Lyngby, Denmark.
Pedersen and Toft, 2005	Pedersen, E.O. and Toft, P. (2005): "Positivisme: erfaringsbaseret viden formuleret i en logisk sprogramme", in Videnskabsteori i Samfunds-videnskaberne (pp. 55-78), Frederiksberg: Roskilde Universitetsforlag.
Pinch and Bijker, 1984	Pinch, T. and Bijker, W.E. (1984): "The social construction of facts and artefacts: Or how the sociology of science and the sociology of technology might benefit each other", in Social Studies of Science, 14(3): 399-441.
Popper, 1934/1980	Popper, K.R. (1980): "The logic of scientific discovery" (trans. of "Logic der forschung", 1934), 10th edition, London: Routledge.
Priestley, 2003	Priestley, M. (2003): "Practical object-oriented design with UML", 2nd edition, New York: McGraw-Hill.
Rasborg, 2005	Rasborg, K. (2005): "Socialkonstruktivismer i klassisk og moderne sociologi", in Videnskabsteori i samfundsvidenskaberne (pp. 349-387), Frederiksberg: Roskilde Universitetsforlag.
Riis, 2003	Riis, J. (2003): "Fremgangsmåde for opbygning, implementering og vedligeholdelse af produktmodeller", PhD dissertation, Department of Manufacturing Engineering and Management, Technical University of Denmark.
Robson, 2002	Robson, C. (2002): "Real world research", 2nd edition, Cornwall: Blackwell Publishing.
Rogoll and Pillar, 2004	Rogoll, T. and Pillar, F. (2004): "Product configuration from the customer's perspective: A comparison of configuration systems in the apparel industry", in Proceedings of International Conference on Economic, Technical and Organizational aspects of Product Configuration Systems (PETO), Lyngby, Denmark.
Raatikainen et al., 2004	Raatikainen, M., Soininen, T., Männistö, T. and Matilla, A. (2004): "A case study of two configurable software product families", in Lecture Notes in Computer Science, 3014: 403-421.
Sabin and Freuder, 1998	Sabin, M. and Freuder, E.C. (1998): "Detecting and resolving inconsistency and redundancy in conditional constraint satisfaction problems", in Web-published papers of the CP'98 Workshop on Constraint Problem Reformulation, Pisa, Italy.
Sabin and Weigel, 1998	Sabin, D. and Weigel, R. (1998): "Product configuration frameworks - A survey", in IEEE Intelligent Systems and Their Applications, 13(4): 42-49.
Salvador and Forza, 2004	Salvador, F. and Forza, C. (2004): "Configuring products to address the customization-responsiveness squeeze: A survey of management issues and opportunities", in International Journal of Production Economics, 91(3): 273-291.
Salvador and Forza, 2007	Salvador, F. and Forza, C. (2007): "Principles for efficient and effective sales configuration design", in International Journal of Mass Customization (IJMASSC), 2(1/2): 114-127.

Salvador et al., 2004	Salvador, F., Rungtusanatham, M. and Forza, C. (2004): "Supply-chain configurations for mass customization", in Production Planning & Control, 15(4): 381-397.
Schmidt, 1984	Schmidt, L.-H. (1984): "Nietzsches 'næse'", in Slagmark, 1(2): 44-62.
Silverman, 2005	Silverman, D: (2005), "Interpreting qualitative data", 2nd edition, London: Sage Publications.
Soininen and Niemela, 1999	Soininen, T. and Niemela, I. (1999): "Developing a declarative rule language for applications in product configuration", in Lecture Notes in Computer Science, 1551: 305-319.
Soininen at al., 1998	Soininen, T., Tiihonen, J., Männistö, T. and Sulonen, R. (1998): "Towards a general ontology of configuration", in Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 12(4): 357–372.
Speel et al., 2001	Speel, P.-H., Schreiber, A., Joolingen, W. and Beijer, G. (2001): "Conceptual models for knowledge-based systems", in Encyclopaedia of Computer Science and Technology, New York: Marcel Dekker Inc.
Studer et al., 1998	Studer, R., Benjamins, V.R. and Fensel, D. (1998): "Knowledge Engineering: principles and methods", in Data & Knowledge Engineering, 25(1-2): 161-197.
Stumptner, 1997	Stumptner, M. (1997): "An overview of knowledge-based configuration", in AI Communications 10(2): 111–125.
Stumptner and Haselböck, 1993	Stumptner, M. and Haselböck, A. (1993): "A generative constraint formalism for configuration problems", in Proceedings of the 3rd Congress of the Italian Association for AI (AI*IA '93) (pp. 302-313), Turin, Italy.
Stumptner at al., 1994	Stumptner, M., A. Haselbock and Friedrich, G. (1994): "COCOS - a tool for constraint-based, dynamic configuration", in Proceedings of the 10th IEEE Conference on AI Applications (CAIA), San Antonio, Texas.
Svensson, 2003;	Svensson, C. (2003): "Mass customization and build to order production - In manufacturing networks", PhD dissertation, Department of Manufacturing Engineering and Management, Technical University of Denmark.
Thyssen, 2003	Thyssen, O. (2003): "Hjørnesten i Niklas Luhmanns systemteori", in Luhmann, 2003, Iagttagelse og Paradoks (pp. 7-43), Danish trans. by H.C. Fink, J. Katlev and O. Thyssen, Copenhagen: Gyldendal.
Tiihonen, 1994	Tiihonen, J. (1994): "Computer-assisted elevator configuration", Masters Thesis, Helsinki University of Technology.
Tiihonen, et al., 1996	Tiihonen, J., Soininen, T., Männistö, T. and Sulonen, R. (1996): "State of the practice in product configuration - A survey of 10 cases in the Finnish industry", in T. Tomiyama, M. Mäntylä and S. Finger (Eds.), Knowledge Intensive CAD (pp.95-114), London: Chapman & Hall.
Tseng at al., 2005	Tseng, H.E., Chang, C.C. and Chang, S.H. (2005): "Applying case-based reasoning for product configuration in mass customization environments", in Expert Systems with Applications, 29(4): 913-925.
Wenneberg, 2002	Wenneberg, S.B. (2002): "Socialkonstruktivisme", Frederiksberg: Samfundslitteratur.
Wright et al., 1993	Wright, J.R., Weixelbaum, E.S., Brown, K., Vesonder, G.T., Palmer, S.R. Berman, J.I. and Moore, H.G. (1993): "A knowledge-based configurator that supports sales, engineering, and manufacturing at AT&T network systems", in AI Magazine, 14(3): 69-80.

Appendix 1: Reflections on the transition from ETO to MC

Reflections on the transition from ETO to Mass Customization

(Proceedings of MCPC 2007, Boston, Oct. 7-10, 2007)

Anders Haug, Klaes Ladeby, Kasper Edwards
{ahaug, kla, ke}@ipl.dtu.dk
Department of Manufacturing Engineering and Management,
Technical University of Denmark,
2800 Kgs. Lyngby, Denmark

Abstract

Most literature discussing the concept of mass customization focuses on cases where companies move from mass production to mass customization. In some literature also ETO (Engineer To Order) companies are claimed to have become mass customizers. However, it can be questioned if these companies conform to popular definitions of mass customizers. This issue raises two questions. Firstly, under which conditions is it reasonable to label ETO companies as mass customizers? Secondly, what are the differences in the characteristics of the transition when mass producers and ETO companies move towards mass customization?

This paper argues that some ETO companies could be labelled as mass customizers although the products are not at prices near mass produced ones. To avoid dilution of the concept of mass customization, while not excluding ETO companies, it is suggested to start out with a broad definition of mass customization under which separate definitions of different kinds of mass customizers are created. The need for specific definitions of different kinds of mass customizers is illustrated by pointing out some of the differences in the transition to mass customization for mass production and ETO companies.

Keywords: Mass customization, personalization, customer co-design

1 Introduction

The term "mass customization" was coined by Davis in the book "Future perfect" from 1987 (Davis, 1987). The general perception of mass customization is to offer customers customized products (goods and services) at prices close to the ones of mass production. The increasing demand for customized products could make it seem that mass customization would be a logical step from mass production for many companies. However, according to some researchers, mass customization has not yet had the impact that many had expected, and it is still much of a niche business (Piller and Ihl, 2002; Zipkin, 2001; Piller, 2004).

Most mass customization literature focuses on cases where companies move from mass production to mass customization, and as a consequence the definitions of mass customization are somewhat one-sided, focusing only on this kind of transition. In some literature also other types of companies are claimed to have become mass customizers (Pine et al., 1995; Duray, 2002; Hvam, 2006; Petersen and Jørgensen, 2005; Steger-Jensen and Svensson, 2004). However, it can be questioned if these companies conform to popular definitions of mass customizers, such as being able to offer products at prices close to mass produced products. In this paper the focus is on one particular kind of company, namely ETO (Engineer To Order) companies, which are the main focus area of a research group at the Centre for Product Modelling (CPM) at the Technical University of Denmark. Since this research group is a part of the mass customization research society it is important to position the type of mass

customizer that this group deals with within the field of mass customization. Therefore, two important questions need to be answered. Firstly, is it reasonable to label ETO companies who create a fixed solution space for the design of customized products as mass customizers or more specifically, under which conditions would this be appropriate? Secondly, what are the differences in the transition characteristics when mass producers and ETO companies move towards mass customization? To emphasise that ETO companies moving towards mass customization do not necessarily become mass customizers in the traditional sense, this paper deliberately uses the phrase "towards mass customization" instead of "to mass customization".

This paper argues that it makes sense to label some ETO companies as mass customizers although the end-products are not at prices near mass produced ones, as many popular definitions of mass customization require. To avoid dilution of the concept of mass customization while not excluding ETO companies it is suggested to start out with a broad definition of mass customization under which more specific definitions of different kinds of mass customizers are created. The need for more specific definitions of different types of mass customizers is illustrated by pointing out some of the major differences between mass production and ETO companies in their transition towards mass customization.

The paper is structured as follows: In section 2 a brief review of definitions of mass customisation is presented and related to ETO companies that move towards mass customization. In section 3 it is discussed if ETO companies can become mass customizers and under which conditions it would be reasonable to describe them as such. In section 4 the difference in the transition from mass producer and ETO to mass customization is discussed. The paper ends with a conclusion in section 5.

2 Mass customization

Naturally, the first definition of mass customization was proposed by the creator of the term, Davis, who defines mass customization as when "the same large number of customers can be reached as in mass markets of the industrial economy, and simultaneously they can be treated individually as in the customised markets of pre-industrial economies" (Davis, 1987). Later, Pine (1993) made an important contribution to the mass customization literature with his "Mass Customization: The new frontier in Business Competition". In this book Pine defines mass customization as "to provide tremendous variety, and individual customization, at prices comparable to standard goods and services”. Another early definition of mass customization is made by Hart (1995), who actually presents two definitions - a visionary definition: "the ability to provide your customers with anything they want profitably, any time they want it, anywhere they want it, any way they want it", and a practical definition: "the use of flexible processes and organizational structures to produce varied and often individually customized products and services at the low cost of a standardized, mass-production system". But not only mass producers can become mass customers according to Pine at al. (1995), who describe how the company Ross Controls, who is from the custom industry, by using CAD and CNC technology together with specialized sales personnel became able to mass customize.

Other literature discusses customization without the "mass". According to Lampel and Mintzberg (1996) the right degree of customization is dependent on the kind of industry a company is part of. They mention two extremes, "mass industries" (manufacturing standardised goods, often in large volumes) and "thin industries" (large degree of customization, often in low volumes). They argue that an important consequence of the shift to what they refer to as "customized standardization" of companies at both ends of the continuum means that customers loose flexibility in one area and gain flexibility in the other area. Hereby, they point out an important distinction between mass production and ETO companies that move towards mass customization, i.e. an increase of product variety and a decrease of product variety respectively.

Gilmore and Pine (1997) identify four distinct approaches to customization, where more than one of these can be applied at the same time: collaborative (dialogue with individual customers to help them articulate their needs, and to make customized products for them); adaptive (offer one standard, but a customizable product that users can alter themselves); cosmetic (present a standard product differently to different customers); and transparent (provide unique goods/services without telling customers explicitly about the customization). Both mass production and ETO companies that move towards mass customization would normally be categorised as "collaborative". However, when ETO

companies move from pure customization towards mass customization this would often be more transparent than when mass production companies move to mass customization. When mass production companies move to mass customization the goal is often to provide more options for the customer, for which reason these options are made very visible to the customer. On the other hand, many companies that move from ETO towards mass customization have a focus on optimizing internal processes (Hvam, 2004; Hvam, 2006; Petersen and Jørgensen, 2005; Steger-Jensen and Svensson, 2004; Hansen et al., 2003). Since customers of ETO products normally expect to get products that are tailored to their needs, the use of a predefined solution space in which the customization takes place may not be communicated to the customers. Similarly, in some cases the movement from ETO towards mass customization is supported by a configuration system, which means that it becomes possible to produce a quote much faster than normal. However, presenting a quote very rapidly could by some customers be perceived as lack of seriousness, why some companies may pretend that specification tasks take more time than they actually do. In such cases the standardization may, therefore, not be very visible to the customer.

Other later and popular definitions also seem to be rooted in mass production, such as the one from Tseng and Jiao (2001): "to deliver goods and services that meet individual customers´ needs with near mass production efficiency". Such a definition does not necessarily exclude the movement to mass customization for ETO companies, depending on whether it is reasonable to label companies in which parts of the products are produced at near mass production efficiency as mass customizers, and depending on what can be labelled as services. This discussion is the focus of the next chapter. Another more recent definition is the one by Silveira et al. (2001), who defines mass customization as relating to "the ability to provide customized products or services through flexible processes in high volumes and at reasonably low costs". While such a definition provides a nice picture of the idea of mass customization, it may be less operational when analysing ETO companies that move towards mass customization, since the provided type of products are often never mass produced, which makes it hard to evaluate what qualifies these products as being produced in high volumes and at reasonably low costs.

Piller (2004) presents a definition of mass customization that offers a higher level of detail than most other such definitions, namely: "Customer co-design process of products and services, which meet the needs of each individual customer with regard to certain product features. All operations are performed within a fixed solution space, characterized by stable but still flexible and responsive processes. As a result, the costs associated with customization allow for a price level that does not imply a switch in an upper market segment". The objective of the article by Piller (2004) is to analyse the recent state of mass customization practice by answering the four basic questions of: "Do customers need customized products?", "If yes, what prevents them from purchasing these offerings?", "Do we have the enabling technologies for mass customization?", and "why do many firms fail during and after the introduction of mass customization?" To answer these questions, Piller makes twelve propositions about mass customization. Piller focuses his discussion on companies that are serving typical "mass" markets, conventionally characterized by made-to-stock and inventory-based distribution systems. For this reason some of Piller's propositions in the same article exclude movements from ETO to mass customization. This is for instance seen in his proposition 8, in which it is stated that customers face risks directly from the customization process. However, in a scenario where an ETO company moves towards mass customization, customers already faced this risk when the company was an ETO company, and moving towards mass customization, if anything, only minimizes the risk. The same goes for proposition 9, in which it is stated that mass customizers need to prevent "mass confusion", which again will not be the case when an ETO company moves towards mass customization, because the solution space is reduced in this case.

Duray et al. (2000) propose a mass customization typology that describes four different approaches to implementing a mass customization capability. The typology takes a basis in the presumption that mass customizers can be identified and classified based on two characteristics, namely: the point in the production cycle when the customer gets involved in the specification of the product, and the type of product modularity employed. This forms a matrix that describes the four archetypes: 1) Fabricators, 2) Involvers, 3) Modularizers, and 4) Assemblers. Although the matrix does a nice job of outlining different approaches to mass customization, it does not provide a distinction between mass customizers that are coming from mass production and ETO, since these two types of companies, at

least in principle, both can be of all of the mentioned four types, and more at the same time. Duray et al. do not mention ETO companies, but distinguish between customized crafted products and standardized mass produced products. Furthermore, Duray (2002) makes three propositions, which are supported by data from 126 mass customizers: P1) Companies practicing mass customization produce non-mass customized products, either standard or custom, in the same plant; P2) Standard and custom product manufacturers adopt distinctly different approaches to mass customization; P3) Companies that adopt approaches to mass customization that most closely resembles the non-mass customized products of the plant will exhibit higher financial performance. Addressing proposition 2, the findings of Duray show that standard producers have higher representation in what she calls Modularizers and Assemblers, while companies which produce more than 50 percent of their products as customized have higher representation in what she calls Fabricators (Designers) and Involvers, i.e. an earlier point of customer involvement.

Some of the described literature in this chapter indicates that a move from ETO to mass customization is possible (Pine et al., 1995; Duray et al., 2000; Duray, 2002). However, the review also showed that definitions of mass customization often seems to be rooted in a move from mass production to mass customization, for which reason it can be unclear whether ETO companies who automate part of their specification process can be labelled as mass customizers.

3 Do ETO companies really become mass customizers?

If being a mass customizer requires the ability to produce products at prices close to mass produced ones, at least in theory, ETO companies can become mass customizers. This, however, would require radical changes e.g. in the form of limiting the product variance, automating the engineering tasks by use of knowledge-based systems, and improvement of manufacturing techniques. The question is whether this is actually what happens when ETO companies are described as mass customizers in literature?

Most literature claiming that ETO companies become mass customizers has a main focus on technology and does not in a detailed manner deal with the business-oriented impact of the mass customization projects (Hvam, 2004, Hvam, 2006, Petersen and Jørgensen, 2005, Edwards and Ladeby, 2005, Steger-Jensen and Svensson, 2004; Hansen et al., 2003). This literature does, therefore, not report whether or not product prices near prices of mass produced products have been achieved. However, the studies to some degree indicate that this is not the case even if product prices may be reduced from the automation of some of the specification tasks. Therefore, it seems that most of such ETO companies do not become true mass customizers in the sense that they are capable of producing customized products at prices close to standard products. The question is: should these ETO companies be labelled as something else or should the traditional definitions be redefined?

If we take the case at Adidas (Moser et al. 2006), which few properly would disagree is a case of mass customization, this is an example of a case where only a small part of the product portfolio that can be customized by the customer. By relating the example of mass customizing only a small part of the product portfolio to ETO companies some interesting conclusions can be made. By breaking the entire design process of e.g. cement plants up into small work packages, is it then possible to completely automate some of them? This would require a predefined solution space and a consistent specification process that allows for involvement of the customer in the design process, but this is actually what companies like F.L. Smidth and GEA Niro do (e.g. Hvam, 2004). Therefore, part of the price of creating the product may be at prices near prices of mass produced products, which implies that at least these products could be labelled as being partly mass customizable.

The main focus of the configuration projects at for instance F.L. Smidth and GEA Niro has been to automate the creation of quotes. Whereas the detailed design processes to a great degree is still carried out by engineers in a traditional manner, the creation of quotations based on customer inputs is more or less completely automated by a product configuration system. The quote can be seen as a service or a product that can be produced at a cost that is near the one of letting a customer choose between different standard solutions. From this perspective it seems reasonable to label these processes as mass customization processes. Perceiving product specifications as a mass customizable product is supported by the fact that some of these kinds of companies do not manufacture the specified products themselves, but only provide the product specifications.

Piller (2004) incisively states that mass customization has become a buzzword, where a major problem is that there is no clear definition and common understanding of the term. Furthermore, Piller points out that if not a common agreement on a definition or understanding of mass customization is reached there is a risk that the field of mass customization will become neither an academic discipline nor a broad strategic concept that is recognized by managers. To help avoid the dilution of the concept of mass customization while not excluding ETO companies, this paper emphasises the importance of making a clear distinction between mass customizers that comes from mass production and custom production without ruling out any of these two kinds of movements. Firstly, this requires a mass customization definition that is broad enough to include both kinds of movements, which again implies a definition with lesser focus on having product prices close to the prices of mass produced products (or to avoid a movement to an upper marked segment). Secondly, there is a need for clear definitions and understanding of different sub-types of mass customizers, for which reason the basic definition of mass customization could be extended by definitions of different kinds of mass customizers. In order to do so it requires a clear idea of the differences between different kinds of mass customizers, which is why the next chapter makes a comparison of mass customizers coming from mass production and ETO.

4 Two different transitions towards mass customization

As mentioned, most mass customization literature focuses on the business consequences of mass customization from a mass production point of view, while literature that deals with how ETO companies become mass customizers most often do this from a merely technological point of view. The one-sided focuses could be explained by the fact that the characteristics of transitions of such companies when becoming mass customizers are very different, for which reason generalisations that are broad enough to include both types could lack substance. To illustrate the difference between the transition from mass production and from ETO to mass customization five areas in which the transition shows different characteristics are described in the following. Others could be mentioned, but the purpose in this context is mainly to illustrate that there are important differences.

1) Product variety: For mass producers to move to mass customization requires that the customers are now allowed to choose different product components or properties, before the product is delivered. On the other hand, an ETO company normally creates a new product for each order, and the challenge when moving to mass customization is to predefine the elements of which the new products can consist, which, obviously, limits the options for the customer. In short, mass producers have the task of encouraging product variety while ETO companies have the task of limiting product variety

2) Customer view: When moving from mass production to mass customization, from the customer's point of view, the increased influence on the design of the product has to have a value, otherwise the possible choices are just confusing or ignoring, i.e. what sometimes is referred to as "mass confusion" (Piller et al., 2005). On the other hand, when an ETO company moves towards mass customization, the creation of a predefined product solution space, obviously, involves the risk that the solution space is not adequately large in order to satisfy the requirements of all customers.

3) Manufacturing costs: In this context manufacturing costs refer to all costs associated to fulfilling an order, including engineering design. Moving from mass production to mass customization implies that the manufacturing task becomes more complex by requiring more planning, a more flexible manufacturing process etc. However, when such tasks can be limited, product prices close to the ones of mass production can be achieved, i.e. what most define as mass customization. Obviously, the opposite is the case when an ETO company moves towards mass customization, in that this implies simplification of the manufacturing process and lower costs per manufactured product.

4) Business purpose: The normal incentive for moving from mass production to mass customization is to make the products offered more attractive to the customers in order to generate or increase sales. In order to be a mass customizer (according to many definitions) the prices of the mass customized products must be close to mass produced ones, which means that if sales are not increased, the investment in becoming a mass customizer would not be returned. As mentioned, it seems that the most important indictment of ETO companies that move towards mass customization is to automate some internal processes. But although an increase of sales is not the main purpose, the effects of the

optimisation could have a sales-increasing effect, i.e. from shorter delivery times, more customer involvement in the design process, being able to manufacture faster etc.

5) Configurator challenge: The design choices of the customers in a scenario where a mass production company becomes mass customizer are normally limited compared to an ETO company that becomes a standardized customizer, and since the focus of mass producers that become mass customizers typically is to increase sales, the user-interface of web-configurators becomes of the highest importance (Rogoll and Piller, 2004; Piller, 2004). On the other hand, ETO products are often hard to standardize to a degree that allows configuration, for which reason the knowledge-base design generally is one of the main challenges, when creating a configuration system for an ETO company that becomes a standardized customizer (Sabin and Weigel, 1998; Hansen et al., 2003; Edwards and Ladeby, 2005).

The five characteristics described are summed up in figure 1.

General characteristics of transition towards mass customization		
	Mass Production to Mass Customization	**Engineering To Order to Mass Customization**
1) Product variety	Increase variety	Limit variety
2) Customer view	Create valuable variety	Maintain adequate variety
3) Manufacturing costs	Slight increase	Decrease
4) Business purpose	Increase sales	Optimise processes
5) Configurator challenge	User interface design	Knowledge base design

Figure 1: Two paths to mass customization

The described differences between the transitions to mass customization from mass production and ETO support the need for having different sub-definitions of mass customization. Such sub-definitions of mass customization could in the case of mass producers and ETO producers be something like: "Typically, the incentive for mass producers to become mass customizers is to allow a customer co-design process while keeping the costs of products comparable to the ones of mass produced, while the incentive for custom producers for pursuing a mass customization strategy is to optimize internal processes by defining fixed solution space in where the customer co-design can take place".

5 conclusions

Most mass customization literature deals with mass producers that move to mass customization. However, some literature deals with another kind of movement, namely when ETO companies moves towards mass customization. This paper pointed out that when ETO companies moves towards mass customization, these do not necessarily become mass customizers in the sense that these are capable of producing products at prices close to if such products had been mass produced. Due to standardization such companies may be able to deliver customized products at prices significantly lower than traditional ETO companies, and hereby, from a product price perspective, be placed somewhere in between ETO and mass customization.

Being a mass customizer does not rule out that the company also creates mass produced products. But what if, instead of a part of the product portfolio, it is part of the separate products that can be mass customized? This pattern can be found in some ETO companies where only some of the product solution space is predefined (i.e. the choices in the early design phases), while detailed design decisions do not take place within a predefined solution space. Since part of the production can be seen as mass customized, such companies can at least be labelled as partly mass customizers. Another suggestion of the paper was that companies could be labelled as mass customizers even if these are not capable of providing customized end-products at prices similar to if these had been mass produced, but on the other hand, if, by the use of standardisation and configuration technology, they are capable of

delivering customized product specifications (such as quotes) at costs close to delivering standard specifications. In this context an important observation is that some ETO companies do not manufacture the physical product themselves, but only create product specifications, which is, therefore, their product.

It is important that the definition of mass customization does not exclude a category of companies. Thus, this paper proposes firstly to define mass customization in a broad sense that does not exclude or indicate that other movements than the one from mass production is possible. Secondly, this definition should be extended by more specific sub-definitions of different kinds of mass customizers. To illustrate this need for more specific definitions of different types of mass customizers some of the important differences between mass customizers that come from mass production and ETO were discussed. The differences discussed in this paper are: if the product variety increases or decreases; if the challenge is to provide valuable or adequate product variety; if the total costs of manufacturing a product increase or decrease; if the main business purpose of the project is to increase sales or optimise internal processes; and if the main configurator challenge is to create user interfaces or the knowledge base of the product configurator.

It is the hope that this paper can contribute to the creation of a common definition and understanding of mass customization, which is a task that still seems to require much research and discussions.

References

Davis, S.M. (1987). *Future Perfect*. Reading, MA: Addison-Wesley Publishing.

Duray, R. (2002). Mass customization origins: mass or custom manufacturing? *International Journal of Operations & Production Management*, Vol. 22, No. 3: 314-28.

Duray, R., Ward, P.T., Milligan, G.W, Berry, W.L. (2000). Approaches to mass customization: configurations and empirical validation. *Journal of Operations Management*, Vol. 18, No. 6: 605–625.

Edwards, K. and Ladeby, K. (2005). Framework for Assessing Configuration Readiness. *In Proceedings of the World Congress on Mass Customization and Personalization 2005 (MCPC 2005)*, Hong Kong, 18-21 Sept. Hong Kong: HKUST.

Gilmore, J. and Pine, B.J. (1997). The Four Faces of Mass Customization. *Harvard Business Review*, Vol. 75, No. 1: 91-101.

Hansen, B., Riis, J. and Hvam, L. (2003). Specification process reengineering: concepts and experiences from Danish industry. *In Proceedings of the 10th ISPE international Conference on Concurrent Engineering: Research and Applications*, Madeira, Portugal, July 26-30. A.A. Balkema Publishers.

Hart, C.W.L. (1995). Mass customization: conceptual underpinnings, opportunities and limits. *International Journal of Service Industry Management*, Vol. 6, No. 2: 36-45.

Hvam, L. (2004). A Multi-perspective approach for the design of Product Configuration Systems - An evaluation of industry applications. *In Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems*, Lyngby, Denmark, 28-29 June. Lyngby, Denmark: Technical University of Denmark, 13-25.

Hvam, L. 2006). Mass customisation of process plants. *International Journal of Mass Customisation*, Vol. 1, No. 4: 445-462.

Lampel, J. and Mintzberg, H. (1996). Customizing customization. *Sloan Management Review*, Vol. 38, No. 1: 21-30.

Moser, K., Muller, M. and Piller, F. (2006). Transforming mass customisation from a marketing instrument to a sustainable business model at Adidas. *International Journal of Mass Customisation*, Vol. 1, No.4: 463-479.

Petersen, T.D. and Jørgensen, K.A. (2005). Product Modelling for Mass Customisation in Global ETO Companies. *In* Blecker, T. and Friedrich, G., eds. *Mass Customization Concepts - Tools - Realization:*

Proceedings of the International Mass Customization Meeting 2005 (IMCM'05), Klagenfurt, Austria, 2-3 June. Berlin: GITO-Verlag, 333-344.

Piller, F. (2004). Mass Customization: Reflections on the State of the concept. *International Journal of Flexible Manufacturing Systems*, Vol. 16, No. 4: 313-334.

Piller, F., Schubert, P., Koch, M. and Möslein, K. (2005). Overcoming Mass Confusion: Collaborative Customer Co-design in Online Communities. *Journal of Computer-Mediated Communication*, Vol. 10, No. 4: article 8.

Piller, F. and Ihl, C. (2002). Mass Customization ohne Mythos. *New Management*, Vol. 71, No. 10: 16-30.

Pine, B.J. (1993). *Mass Customization: The New Frontier in Business Competition*. Boston: Harvard Business School Press.

Pine, B.J., Peppers, D. and Rogers, M. (1995). Do you want to keep your customers forever? *Harvard Business Review*, Vol. 73, No. 2: 103-114.

Rogoll, T. and Piller, F. (2004). Product configuration from the customer's perspective: A comparison of configuration systems in the apparel industry. *In Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems*, Lyngby, Denmark 28-29 June. Lyngby, Denmark: Technical University of Denmark, 177-198.

Sabin, D. and Weigel, R. (1998). Product Configuration Frameworks - A survey. *IEEE Intelligent Systems & Their Applications*, Vol. 13, No. 4: 42-49.

Silveira, G.D., Borenstein, D. and Fogliatto, F.S. (2001): Mass customization: Literature review and research directions. *International Journal of Production Economics*, Vol. 72, No. 1: 1-13.

Steger-Jensen, K. and Svensson, C. (2004). Issues of mass customisation and supporting IT-solutions. *Computers in Industry*, Vol. 54, No. 1: 83-103.

Tseng, M. and Jiao, J. (2001). Mass Customization. *In* Salvendy, G. ed. *Handbook of Industrial Engineering*, 3rd edition. New York: Wiley: 684-709.

Zipkin, P. (2001). The Limits of Mass Customization. *Sloan Management Review*, Vol. 42, No. 3: 81-87.

Appendix 2: List of publications

Journal articles

[1] Haug, A., Degn, A., Poulsen, B., and Hvam, L. (2007): "A prototype of a documentation system that supports the development and maintenance of product configuration systems", WSEAS Transactions on Information Science and Applications, Issue 5, Vol. 4.

[2] Haug, A. and Hvam, L. (2006): "CRC-cards to support development and maintenance of product configuration systems", submitted to International Journal of Mass Customization on request from editor, 16/10-2006 (under review).

[3] Haug, A. and Hvam, L. (2007): "A comparative study of two graphical notations for the development of product configuration systems", International Journal of Industrial Engineering, Vol. 14, No. 2.

[4] Haug, A. and Hvam, L. (2007): "The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems", International Journal of Mass Customization, Issue 1/2, Vol. 2.

[5] Haug, A., Ladeby, K., and Edwards, K. (2007): "From Engineer-To-Order to Mass Customization", submitted to Management Research News on request from editor, 20/8 2007 (under review).

Peer reviewed conference papers

[6] Haug, A. and Hvam, L. (2005): "Developing 3D configuration systems for manufacturers of complex building components", in Mass Customization, Concepts - Tools - Realization (Proceedings of IMCM 2005, Klagenfurt, Austria, June 2-3, 2005), GITO-Verlag, Berlin.

[7] Haug, A. and Hvam, L. (2005): "Product analysis as a basis for building product configuration systems", in Proceedings from the 3rd Interdisciplinary World Congress on Mass Customization and Personalization (MCPC '05), Hong Kong, Sept. 18-21, 2005.

[8] Haug, A. and Hvam, L. (2006): "The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems", in Customer Interaction and Customer Integration (Proceedings of the Joint Conference IMCM '06 & PETO '06, Hamburg, Germany, June 22-23, 2006), GITO-Verlag, Berlin.

[9] Haug, A. and Hvam, L. (2006): "CRC-cards for the development and maintenance of product configuration systems", in Customer Interaction and Customer Integration (Proceedings of the Joint Conference IMCM '06 & PETO '06, Hamburg, Germany, June 22-23, 2006), GITO-Verlag, Berlin.

[10] Haug, A. and Hvam, L. (2006): "Merging models with different perspectives on product configuration knowledge", in Research in Interactive Design, Volume 2 (Proceedings of Virtual Concept, Cancun, Mexico, Nov. 26 - Dec. 1, 2006), Springer-Verlag, France.

[11] Haug, A., Degn, A., Poulsen, B., and Hvam, L. (2007): "Creating a documentation system to support the development and maintenance of product configuration systems", in Proceedings of the 2007 WSEAS International Conference on Computer Engineering and Applications, Queensland, Australia, Jan. 17-19, 2007.

[12] Haug, A. and Hvam, L. (2007): "Tacit knowledge in configuration projects", in Innovative Processes and Products for Mass Customization (Proceedings of the Joint Conference IMCM '07 & PETO '07, Hamburg, Germany, June 21-22, 2007), GITO-Verlag, Berlin.

[13] Haug, A., Ladeby, K., and Edwards, K. (2007): "Reflections on the transition from ETO to Mass Customization", in Proceedings of MCPC 2007, Boston, Oct. 7-10, 2007.

[14] Haug, A. and Hvam, L. (2007): "Product Structured Class Diagrams to support the development of Product Configuration Systems", in Proceedings of MCPC 2007, Boston, Oct. 7-10, 2007.

[15] Haug, A. and Hvam, L. (2007): "A classification of the information that domain experts do and do not provide in configuration projects", in Proceedings of the 12th annual international conference on Industrial Engineering Theory, Applications & Practice, Cancun, Mexico, Nov. 4-7, 2007.

[16] Edwards, K., Ladeby, K., and Haug, A. (2007): "Measuring process and organizational consistency - A necessary step before implementing configuration systems", in Innovative Processes and Products for Mass Customization (Proceedings of the Joint Conference IMCM '07 & PETO '07, Hamburg, Germany, June 21-22, 2007), GITO-Verlag, Berlin.

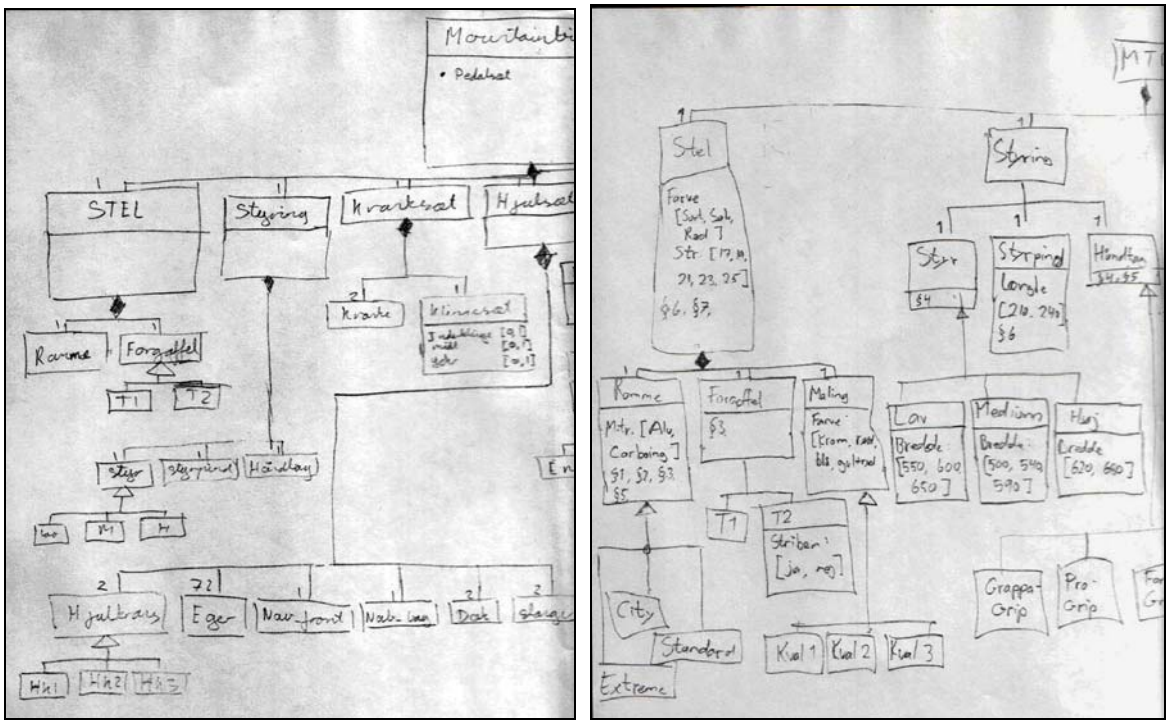
[17] Ladeby, K. Edwards, K., and Haug, A. (2007): "Typology of Product Configuration Systems", in Innovative Processes and Products for Mass Customization (Proceedings of the Joint Conference IMCM '07 & PETO '07, Hamburg, Germany, June 21-22, 2007), GITO-Verlag, Berlin.

Appendix 3: Contribution to appended papers

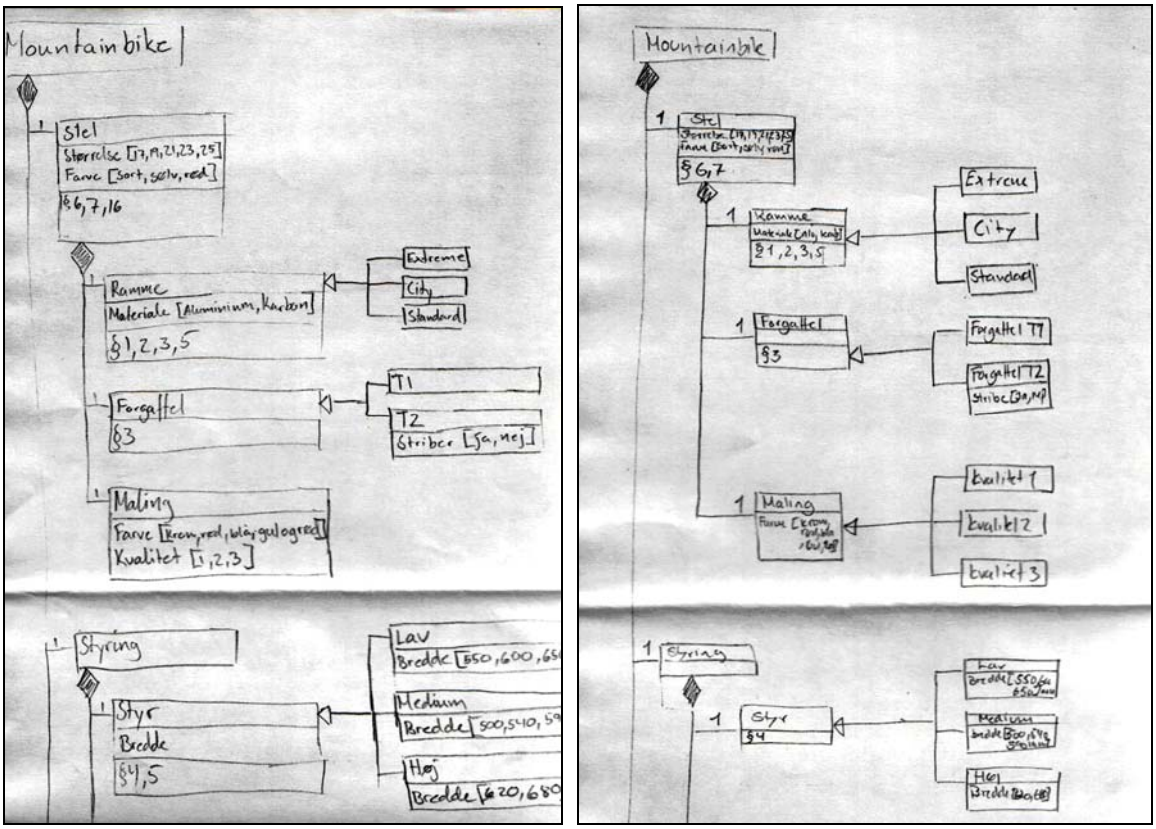
Paper		Contribution
A1	Haug, A. and Hvam, L. (2007): "Tacit knowledge in configuration projects", in Innovative Processes and Products for Mass Customization (Proceedings of the Joint Conference IMCM'07 & PETO'07, Hamburg, Germany, June 21-22, 2007), GITO-Verlag, Berlin.	A. Haug: 100 % L. Hvam: Approval of manuscript
A2	Haug, A. and Hvam, L. (2007): "A classification of the information that domain experts do and do not provide in configuration projects", in Proceedings of the 12th annual international conference on Industrial Engineering Theory, Applications & Practice, Cancun, Mexico, Nov. 4-7, 2007.	A. Haug: 100 % L. Hvam: Approval of manuscript
B1	Haug, A. and Hvam, L. (2005): "Product analysis as a basis for building product configuration systems", in Proceedings from the 3rd Interdisciplinary World Congress on Mass Customization and Personalization (MCPC'05), Hong Kong, Sept. 18-21, 2005.	A. Haug: 100 % L. Hvam: Approval of manuscript
B2	Haug, A. and Hvam, L. (2007): "A comparative study of two graphical notations for the development of product configuration systems", International Journal of Industrial Engineering, Vol. 14, No. 2.	A. Haug: 100 % L. Hvam: Approval of manuscript
B3	Haug, A. and Hvam, L. (2007): "Product Structured Class Diagrams to support the development of Product Configuration Systems", in Proceedings of MCPC 2007, Boston, Oct. 7-10, 2007.	A. Haug: 100 % L. Hvam: Approval of manuscript
B4	Haug, A. and Hvam, L. (2006): "Merging models with different perspectives on product configuration knowledge", in Research in Interactive Design, Volume 2 (Proceedings of Virtual Concept, Cancun, Mexico, Nov. 26 - Dec. 1, 2006), Springer-Verlag, France.	A. Haug: 100 % L. Hvam: Approval of manuscript
C1	Haug, A. and Hvam, L. (2007): "The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems", International Journal of Mass Customisation, Issue 1/2, Vol. 2.	A. Haug: 100 % L. Hvam: Approval of manuscript
C2	Haug, A. and Hvam, L. (2006): "CRC-cards for the development and maintenance of product configuration systems", in Customer Interaction and Customer Integration (Proceedings of the Joint Conference IMCM'06 & PETO'06, Hamburg, Germany, June 22-23, 2006), GITO-Verlag, Berlin.	A. Haug: 100 % L. Hvam: Approval of manuscript
C3	Haug, A., Degn, A., Poulsen, B., and Hvam, L. (2007): "Creating a documentation system to support the development and maintenance of product configuration systems", in Proceedings of the 2007 WSEAS International Conference on Computer Engineering and Applications, Queensland, Australia, Jan. 17-19, 2007.	A. Haug: 70 % A. Degn: 20% B. Poulsen: 10% L. Hvam: Approval of manuscript

Joint author statements can be acquired upon request.

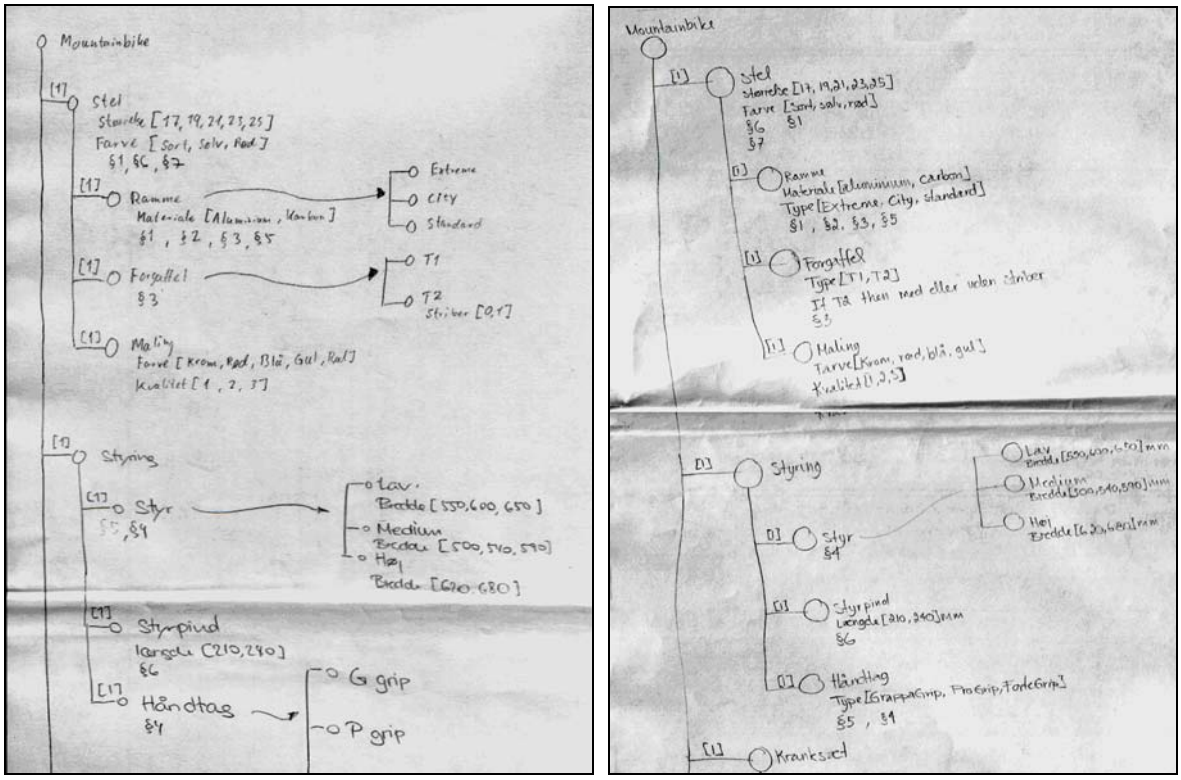
Appendix 4: Usability experiment



Extract from two groups using Class Diagrams



Extract from two groups using PSCDs



Extract from two groups using PVMs

Appended papers

The appended papers have been reformatted in order to improve the readability (mainly the font size of body text). The contents of the papers have not been changed.

Paper A1, copyrighted 2007 by:	IMCM & PETO: International Mass Customization Meeting & International Conference on Economic, Technical and Organisational Aspects of Product Configuration Systems
Paper A2, copyrighted 2007 by:	IJIE: International Journal of Industrial Engineering
Paper B1, copyrighted 2005 by:	MCPC: World Congress on Mass Customization and Personalized Customization
Paper B2, copyrighted 2007 by:	IJIE: International Journal of Industrial Engineering
Paper B3, copyrighted 2007 by:	MCPC: World Congress on Mass Customization and Personalized Customization
Paper B4, copyrighted 2006 by:	Virtual Concept, International Meeting on Interactive Design and Manufacturing Methodologies, Tools and Processes
Paper C1, copyrighted 2007 by:	IJIE: International Journal of Mass Customization
Paper C2, copyrighted 2006 by:	IMCM & PETO: International Mass Customization Meeting & International Conference on Economic, Technical and Organizational Aspects of Product Configuration Systems
Paper C3, copyrighted 2007 by:	WSEAS: The World Scientific and Engineering Academy and Society

Paper A1

Tacit knowledge in configuration projects

Haug, A. and Hvam, L. (2007): "Tacit knowledge in configuration projects", in Innovative Processes and Products for Mass Customization, GITO-Verlag, Berlin (Proceedings of the Joint Conference IMCM'07 & PETO'07, Hamburg, Germany, June 21-22, 2007).

Tacit knowledge in configuration projects

Anders Haug and Lars Hvam

Abstract

Product configurators are software systems that support the specification of customized products by allowing the users to define products, while restricting how different components and properties may be combined. When creating a product configurator, often one of the greatest tasks is to represent the product knowledge of which the knowledge base of the product configurator should consist. One of the main challenges of representing domain knowledge is that much relevant knowledge often resides in the heads of domain experts. A term often applied to refer to difficult accessible knowledge in configuration literature is *tacit knowledge*. However, since there are many different views on what tacit knowledge is, and since the existing literature on product configuration seldom provides any definitions or examples of tacit knowledge, it is unclear what the term is intended to mean in the product configuration literature. In order to clarify the meaning of tacit knowledge this paper investigates existing definitions from knowledge management literature. Based on this, the paper argues that tacit knowledge may not be particularly relevant when describing the difficulties in representing domain knowledge in configuration projects and that the distinction between tacit and explicit knowledge can be misleading.

Keywords

Product configuration, tacit knowledge, knowledge acquisition

1 Introduction

Product configurators (or product configuration systems) are software systems that support the specification of customized products by allowing the users to define products, while restricting how different components and properties may be combined. In several cases the use of product configurators has produced a range of benefits, such as: shorter lead times, improved quality of product specifications, preservation of knowledge, use of fewer resources for specifying products, optimized products, less routine work, improved certainty of delivery, and less time needed for training new employees (Hvam, 2004; Hvam et al., 2006; Riis, 2003; Forza & Salvador, 2002; Forza et al., 2005).

When creating product configurators in cases where ETO (Engineering To Order) companies choose to apply configurator-supported product specification, one of the greatest tasks is to represent the product knowledge of which the knowledge base of the product configurator should consist (Sabin & Weigel, 1998; Hansen et al., 2003; Edwards & Ladeby, 2005). One of the reasons why it is difficult to represent domain knowledge is that much of the relevant knowledge is often not accessible in a well-structured form.

In product configuration literature the term *tacit knowledge* is often applied to refer to difficult accessible knowledge. However, it can be difficult to know what the term is intended to cover in these contexts, since there is no general agreement on what tacit knowledge is (Gourlay, 2006), and configuration literature seldom defines or gives examples of what the term is supposed to refer to. The vague definitions of tacit knowledge in product configuration research obviously represent a problem in relation to building on this research. To address this problem, this paper investigates existing definitions from knowledge management literature and discusses the implications on product configuration research from using more strict definition of tacit knowledge. Based on this, the paper argues that tacit knowledge may not be particularly relevant when describing the difficulties in representing domain knowledge in configuration projects and that the distinction between tacit and explicit knowledge can be misleading.

The paper is structured as follows: Firstly, in section 2, an introduction to tacit knowledge is given. Next, in section 3, an investigation of the use of the term *tacit knowledge* in configuration literature is presented. For a deeper look into the meaning of tacit knowledge, section 4 provides a brief summery of views on tacit knowledge in knowledge management literature. In section 5, the paper analyses the implications of the existing use of the term *tacit knowledge* in product configuration literature and proposes a more strict definition. The paper ends with a conclusion in section 6.

2 Tacit knowledge

Before engaging in a discussion of tacit knowledge, it may seem appropriate to give a definition of the word *knowledge* itself. However, debates about the nature of knowledge have been part of the philosophical literature since the classical Greek period, and still are. Philosophers recognise three main kinds of knowledge: practical knowledge (knowing-how), knowledge of people, places and things (knowledge by acquaintance) and propositional knowledge (knowing-that or factual knowledge) (Bernecker & Dretske, 2000). Propositional knowledge takes the form of "S knows that P", where P is a declarative sentence that expresses some proposition. The traditional epistemology is primarily targeted at propositional knowledge.

A classical understanding of knowledge, which can be traced back to Plato's "Theaetetus" (c.400 BC) and Kant's "Critique of pure reason" (from 1781), is a definition of knowledge as "justified true beliefs" (Bernecker & Dretske, 2000). Although many still use this tripartite account of knowledge, it has also

been criticised from different angles. For instance by Gettier (1963), who counter-exampled this definition of knowledge, which implied that many epistemologists thought that he had destroyed the long-standing tradition regarding correct analysis of knowledge (Bernecker & Dretske, 2000). Addressing the problem of creating clauses for what is knowledge in "Foundations of social sciences" from 1944, Otto Neurath points to the paradox that epistemology faces, namely a problem of circularity. Any epistemology presupposes knowledge of the conditions in which knowledge takes place, but since science cannot be used in order to ground the legitimacy of science, there are no secure foundations from which we can begin any consideration of our knowledge of knowledge. Therefore, rather what we have are competing philosophical assumptions (Johnson & Cassel, 2001).

Although many have proposed general clauses for how scientific knowledge should be obtained, it seems that different perceptions of what knowledge is work in different contexts. For knowledge-based theories of the firm Spender (1998) concludes that useful theories are less theories of objective entities than sets of contextualised heuristics guiding managers' interventions in their organisations as quasi-autonomous systems. Spender therefore argues for a pluralist epistemology. Epistemological pluralism rejects the notion of a single reference system in which we can establish truth and hereby admits multiple forms of approach, evidence and reasoning (Spender, 1998). This pluralism implies that several knowledge systems can be applicable in a specific context, which gives the researcher a choice when carrying out analysis.

A commonly applied way of classifying knowledge is the distinction between tacit and explicit knowledge. The general definition of explicit knowledge is knowledge that has been (or can be) codified and can readily be communicated to others. However, it can be questioned whether pure explicit knowledge actually exists or such knowledge would merely be information (Stenmark, 2002), just as it can be claimed that even the most explicit kind of knowledge is underlain by tacit knowledge (Tsoukas, 2003).

Only toward the 1980s literature about practical knowledge had a significant impact on the development of society, where some of the most important contributions are from Ludwig Wittgenstein, Gilbert Ryle, and Michael Polanyi (Gustavsson, 2001). In 1949, Gilbert Ryle published the book "The concept of the mind", which were to be a central basis for the discussion of practical knowledge (Gustavsson, 2001). The central term that Ryle introduces is "knowing how", which is put in contrast to the traditional form of knowledge, i.e. "knowing that". By using *knowing* instead of *knowledge*, Ryle emphasizes that knowledge has the character of being an activity (Ryle, 1949).

With the book "Personal knowledge" from 1958, Michael Polanyi created one of the most import contributions regarding the understanding of tacit knowledge. Tacit knowledge is generally used to characterize knowledge that a person is not capable of articulating, and according to Polanyi (1962) all knowledge has a tacit dimension. To illumine the concept of tacit knowledge, Polanyi (1962) argues that the principle of how a cyclist keeps his balance is not generally known. Obviously, rules for how to keep the balance can be observed, i.e. that when a cyclist starts to fall he turns the handlebars, resulting in a centrifugal force that offsets the gravitational force, dragging him down etc. However, this does not tell us how to ride a bicycle and can only be guiding information. This point is summed up by Polanyi's famous quote "we can know more than we can tell" (1966). Polanyi's use of the term *tacit knowledge* most often refers to subconscious processes rather than static structures, captured in Polanyi's words "knowledge is an activity which would be better described as a process of knowing" (Polanyi, 1969).

3 Tacit knowledge in configuration literature

When going through literature that deals with product configuration, one often comes across the term *tacit knowledge*. Based on experience gained from 12 case studies of product configuration projects, Møldrup and Møller (2004) focus on the employment of a change management perspective in product configuration projects. They claim that experts, although well-meaning, often provide information that is "incomplete, inconsistent or inaccurate due to forgetfulness, assumptions that further elaborations are not necessary or the difficulties in making tacit knowledge explicit". In one of the cases of a study by Pedersen and Edwards (2004) they claim that much of the knowledge of the ones doing calculations for tenders is tacit, and that information about products that are complex to modularise often is in the form of tacit knowledge or non-specified knowledge. Pedersen and Edwards furthermore mention a process of making information in the form of tacit knowledge into explicit knowledge. In his thesis about industrial variant specification systems, Hansen (2003) describes tacit knowledge as having a personal quality, for which reason it is hard to formalize and communicate. Hansen discusses varying needs for making tacit (used interchangeably with implicit) specifications explicit, just as tacit knowledge is perceived as something that can be made explicit. Edwards and Ladeby (2005) use the term *tacit knowledge* about knowledge that is applied when a sales person or production employee configures a product. They claim that this knowledge is most often tacit and should be made explicit in order to develop a configurator. They furthermore state that tacit process choices often are based on historical coincidence or forgotten reasons, and claim that through interviews it can be determined whether product knowledge is tacit or explicit. Tiihonen et al. (1998) present a method for managing and modelling product families as configurable products. They mention that one of the deliverables when developing product configurators is an explicit configuration model, and state that "in relatively common industrial practice the configuration models are only implicitly present as tacit knowledge in product experts' minds or in varied documents". Based on a literature study in the context of investigating potential competitive advantages of introducing product configurators for mass customization/tailoring companies, Skjevdal (2005) mentions the advantage of "externalization of tacit knowledge". Other literature that deals with product configuration applies the term *tacit knowledge* without providing much more indication of its meaning than it is some kind of non-explicit knowledge (e.g. Hvam & Malis, 2001; Frutos et al., 2004; Schwarze & Schönsleben, 1996; Heiskala et al., 2005, Hvam et al., 2006).

Based on this somewhat superficial study of the application of the term *tacit knowledge* in product configuration literature, it seems that definitions of the term in such contexts are very rare. Roughly speaking, it seems that some of the investigated literature applies the term *tacit knowledge* merely for describing knowledge that is non-explicit (or unknown), i.e. a remainder category. Also, in the investigated literature the general idea is that tacit knowledge can be made explicit, or at least this is what the chosen formulations indicate. Obviously, the many statements of the sort "making tacit knowledge explicit" may be meant in the sense that "making some explicit descriptions that can simulate some tacit knowledge". However, if this be the case, the latter formulation would have been more accurate.

4 Tacit knowledge in knowledge management literature

To look deeper into the phenomenon of tacit knowledge the focus of this paper is turned from product configuration towards a field that have focused more on analysing and discussing the concept, namely knowledge management. Although the concept of tacit knowledge has been widely applied and discussed in knowledge management literature, a general agreement on the definition or correct application of the term does not exist.

On a basic level tacit knowledge is most often defined as being individual and highly difficult or even impossible to articulate (e.g. Newell at al., 2002; Nonaka et al., 2000; Wiig, 1993, Hitt et al., 2000). Also the term *know-how* is often applied to describe tacit knowledge. About the distinction between tacit and explicit knowledge, Grant (1996) suggests that while "explicit knowledge is revealed by its communication, tacit knowledge is, on the other hand, revealed through its application". In knowledge management literature Polanyi is often credited for his distinction between tacit and explicit knowledge. However, some researchers argue that he has been misunderstood, and that his ideas may not be particularly relevant for understanding tacit knowledge in organisations (Gourlay, 2004; Tsoukas, 2003).

The study by Collins (2001) is one of the most thorough investigations of tacit knowledge. Collins takes a basis in the fact that some scientists can do certain experiments while others cannot and points out that one of the reasons can be that the unsuccessful scientists lack tacit knowledge. In this context, Collins defines tacit knowledge as "knowledge or abilities that can be passed between scientists by personal contact but cannot be, or has not been, set out or passed on in formulae, diagrams, or verbal descriptions and instructions for action".

In a review of empirical studies of tacit knowledge, Gourlay (2006) describes six ambiguities concerning the nature of tacit knowledge: 1) some regard tacit knowledge as being only individual, while others see it as being both individual and collective; 2) in general tacit knowledge is perceived as being acquired through experience. However, some suggest that we are biologically predisposed towards certain kinds of tacit knowledge; 3) while there is widespread agreement that tacit knowledge is acquired through personal contact and observation, some claim that it is acquired with little help from others; 4) tacit knowledge is claimed to be essential for competent performance, but on the other hand it is also claimed to be the basis of defensive routines and contain wrong theories; 5) tacit knowledge is by many regarded as the source of innovative ideas, while some claim that tacit knowledge manifested in traditions is a conservative rather than an innovative force; and 6) some claim that tacit knowledge can be turned into explicit knowledge, while others claim that this cannot be done. When narrowing the focus to look at what tacit knowledge is, rather than how it emerges or its impact, Gourlay's (2006) review of research reveals that the term *tacit knowledge* has been applied in three distinct ways, namely: about knowledge that could be articulated, in cases where only feelings to tacit knowledge was claimed, and in cases where there was evidence of an action or behaviour of which the actors could not give account. Based on these different kinds of use of the term, Gourlay argues that knowledge than can be readily articulated, should not be classified as tacit knowledge, although it might have been tacit at the time it was used. Gourlay further argues that if unobservable behaviours is admitted, in particular people's claims to have thoughts and feelings, there would be no limit to what could count as tacit knowledge. For the sake of clarity of communication, Gourlay proposes that the term *tacit knowledge* should only be used about the third category of empirical use, namely inarticulable knowledge. Gourlay also suggests that the issue of if tacit knowledge can be made explicit is reframed in terms of whether or not functionally equivalent descriptions can be made. A similar definition have been made by Cook and Brown (1999), who argue that tacit knowledge cannot be turned into explicit knowledge or the other way around, so if one uses tacit knowledge and thereby gain explicit knowledge, it is not that the tacit knowledge has been made explicit, just that explicit knowledge has been gained. The idea of tacit knowledge being something that cannot be turned into explicit knowledge is backed up by others (e.g. Tsoukas, 2003; Collins, 2001; Ambrosini & Bowman, 2001). On the other hand, other researchers claim that it is sometimes possible to turn tacit knowledge explicit, but it is often very difficult (e.g. Nonaka & Takeuchi, 1995; Choo et al., 2000; Newell et al., 2002; Spender, 1996).

5 Discussion

As mentioned, it seems that in general the purpose of using the term *tacit knowledge* in product configuration literature is to describe some sort of knowledge that it is difficult to deal with, while it is unclear exactly what the term refers to, since definitions of the term are rarely provided, nor given well-described examples of what the term refers to in a given context. This raises two questions. Firstly, what the implications of the present use of the term in product configuration literature are? And secondly, whether or not the term should be applied in a product configuration context at all?

Roughly speaking, the present use of the term *tacit knowledge* in product configuration literature generally does not tell much more than tacit knowledge is knowledge that is not represented in known explicit representations. If, in a configuration context, the knowledge of a company is described in terms of explicit and tacit knowledge, then, obviously, what is not explicit must be tacit. Knowledge needs to be known in order to be able to classify it as being explicit, which implies that what in configuration literature is described as tacit knowledge in principle could include: inarticulable knowledge, not yet articulated explicit knowledge, knowledge unknown to the observer, forgotten explicit knowledge, and not yet created knowledge. Besides tacit and explicit knowledge, also the term *non-specified knowledge* was found in the investigated literature. However, it is unclear whether this refers to tacit knowledge, explicit knowledge not yet written down and/or not yet existing knowledge.

The implication of the use of the term *tacit knowledge* in product configuration literature is illustrated in table 1, and subsequently discussed.

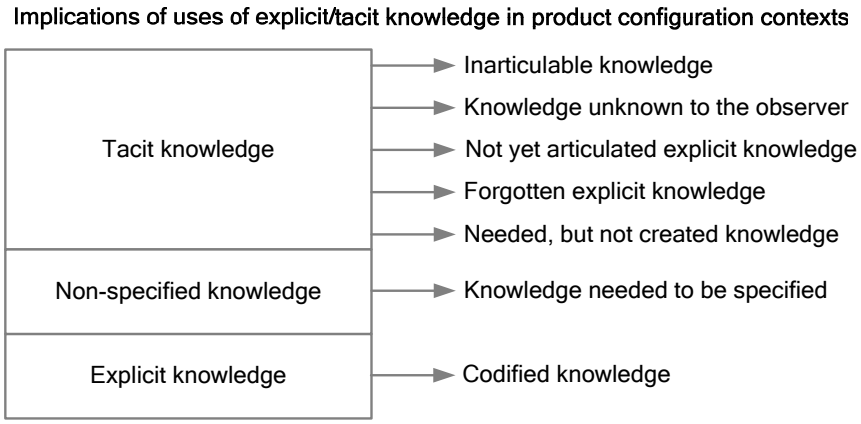


Figure 1: Implication of the uses of tacit knowledge in product configuration literature

The first possible meaning of tacit knowledge in figure 1 is inarticulable knowledge. Inarticulate knowledge is knowledge that allows a person to perform certain actions that the person is incapable of articulating. That this should be perceived as a form of tacit knowledge should be beyond discussion.

Explicit knowledge that is unknown to an observer could for instance be documents that are possessed by a domain expert without a knowledge engineer (observer) knowing about it. It should also be clear that the ignorance of the observer is not an argument for labelling this kind of knowledge as being tacit knowledge.

Unarticulated explicit knowledge could for instance be rules created by a domain expert in his/her mind which he/she consciously applies without ever having articulated these. This is, therefore, a kind of knowledge that is codified and can readily be communicated. Such unexpressed explicit knowledge should not be confused with tacit knowledge.

The term *forgotten explicit knowledge* could, for instance, be applied in a situation where a domain expert takes specific decisions, when facing certain kinds of problems, but where the domain expert cannot remember (justify) why he/she takes these decisions, although this at some point was stated explicitly. This kind of "knowing why" could in some configuration projects be needed in order to create a generic model of the expertise of the domain expert. However, it can be discussed if it makes sense to say that someone tacitly know why they do something, since it could be impossible to determine if this is actually the case, as opposed to know-how, which is something that can be demonstrated.

Another possible type of forgotten explicit knowledge is in a situation where a domain expert is capable of creating some kind of product specification but cannot tell how it is done, although the domain expert was originally explicitly told how. The tacit knowledge that is applied seems to be something else than what was explicitly stated originally, which can be illustrated by the earlier mentioned bicycle example of Polanyi, where what can be explicitly stated about how to bicycle differs from the know-how that is acquired when learning to bicycle. So instead of explicit knowledge being converted into tacit knowledge, rather new tacit knowledge is created while the explicit instructions may be forgotten. This implies that if explicit knowledge is inferred based on observing the actions of the domain expert, this could easily differ from the original explicit knowledge. From this view, forgotten explicit knowledge is knowledge that does not exist until it is recalled or recreated, no matter if it has been the source of some tacit knowledge or not.

Needed, but not yet created knowledge could for instance be non-existent rules that are needed for the creation of the knowledge base of a product configurator. Since something, obviously, is not knowledge before it exists, there should be a clear distinction between tacit knowledge and the non-existing knowledge that is needed in order to create a product configurator.

The vagueness of the meaning of tacit knowledge in product configuration literature can be problematic, and in order to avoid misunderstandings a clear definition of the term *tacit knowledge* is required. This paper proposes the use of a more strict definition, which is closer to how the term was used by Polanyi. Therefore, there should be a clear distinction between the tacit knowledge that a domain expert actually applies and what is represented in a knowledge representation during knowledge acquisition. Tacit knowledge should not be seen as something that can be made explicit, and a knowledge representation created in order to describe some tacit knowledge is rather what the domain expert or an observer has inferred when trying to explain the actions of the domain expert. This view on tacit knowledge also supports the suggestion of Gourlay (2006) only to use the term *tacit knowledge* about inarticulable knowledge.

As the use of the term *tacit knowledge* in product configuration literature is applied for indicating some kind of difficult knowledge, a division of knowledge based on how the different kinds of knowledge can be dealt with seems more appropriate. In figure 2 is shown how the types of knowledge presented in figure 1 could be approached when creating a basis for developing a product configurator.

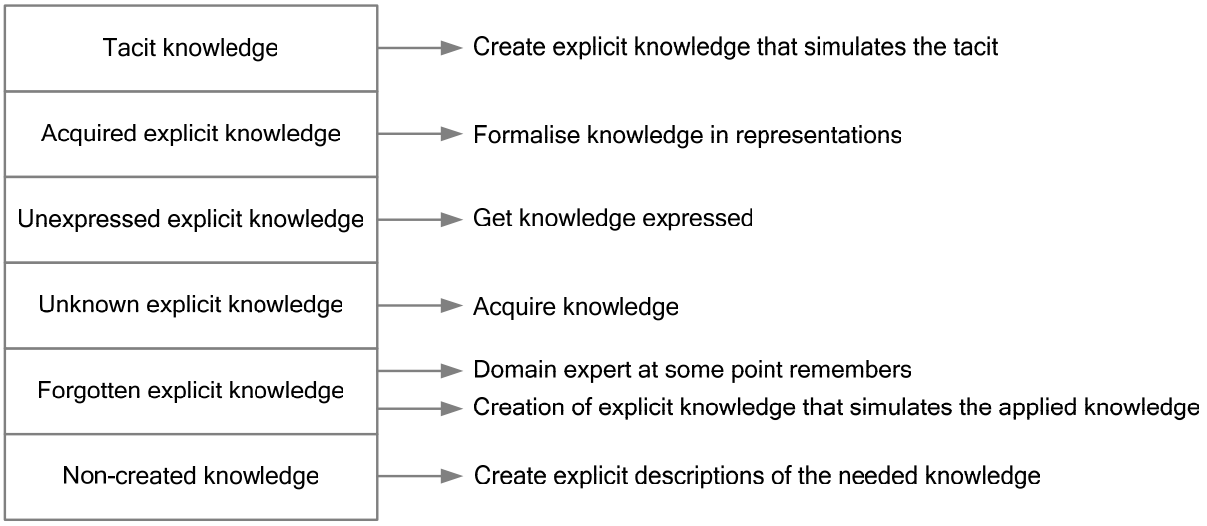


Figure 2: Knowledge engineering approaches towards different forms of knowledge

One of the problems faced when focusing too much on tacit knowledge as some sort of difficult knowledge in a product configuration context is that the amount of tacit knowledge is not necessarily particularly relevant, when considering a configuration project in a company. In a product configuration context, where inarticulable know-how of a domain expert is interesting, it must be presumed that the domain expert is at least capable of telling what is done in specific situations, i.e. what the domain expert can infer. This means that the task of describing the choices made by a domain expert based on particular design requirements should not be a problem, although it may be time-consuming to describe the entire solution space. Since these observable connections between input and output often would be adequate in order to create a configurator, and as the connections may not be very hard to infer, tacit knowledge does not necessarily represent a big problem in a configuration project. On the other hand, it cannot be presumed that explicit knowledge is unproblematic by definition. For instance, in some cases explicit definitions can contradict each other, when different domain experts have different perceptions of what is "the right way of doing things". In other cases the explicit definitions can be problematic if the knowledge engineer does not understand them because of missing product expertise. So, all in all, it seems that the distinction between tacit and explicit knowledge is not the most useful way of describing the relevant knowledge of a company in a product configuration context.

6 Conclusions

In this paper a study of the application of the term *tacit knowledge* in product configuration literature showed that the term is often applied to describe some sort of non-explicit knowledge, which is difficult to deal with. However, it is hard to know exactly what the term is referring to in the existing literature, since the provided definitions of tacit knowledge are very sparse, just as well-described examples of what this tacit knowledge actually represents were not found. Obviously, this indistinctness of the meaning of the term tacit knowledge implies that it can be difficult to build on existing literature.

A brief look into knowledge management literature, which is a field in which many efforts have been made to discuss the concept of tacit knowledge, revealed that within this field there are great ambiguities concerning how tacit knowledge should be perceived. One of the most important ambiguities in a product configuration context is whether tacit knowledge can be turned explicit or not. This paper argues that it in a product configuration context would make most sense only to apply the term *tacit knowledge* about inarticulable knowledge. Firstly, it seems that this kind of use is more in line with what Polanyi had in

mind when he defined the concept. Secondly, the consequence of making the definition of tacit knowledge too broad is that the term would loose its relevance, as it does not tell much about the nature of the knowledge in focus. The conclusion of this paper is, therefore, that in a product configuration context tacit and explicit knowledge should be seen as two very different concepts, which is why tacit knowledge cannot be converted into explicit knowledge or the other way around. Therefore, what actually happens, when it is claimed that tacit knowledge is made explicit, is that new explicit knowledge is created, based on what can be inferred by observing the tacit knowledge in action.

Having drawn a sharp line between what should be classified as tacit knowledge and what should not, this paper argues that the distinction between tacit and explicit knowledge may not be a particular useful way of dividing knowledge in a configuration context, since the fact that some knowledge is tacit does not necessarily represent a difficulty in regard to creating a configuration system, just as explicit knowledge by definition cannot be considered to be easy to convert into a model in the knowledge base of a product configurator. Finding a more suitable way of describing knowledge in product configuration projects is, therefore, a problem for future research.

References

Ambrosini, V. & Bowman, C. (2001): Tacit knowledge: Some suggestions for operationalization. *Journal of Management Studies*, 38(6), 811-829.

Bernecker, S. & Dretske, F. (2000): *Knowledge: Readings in Contemporary Epistemology* (pp. 3-6). New York: Oxford University Press.

Choo, C.W., Detlor, B. & Turnbull, D. (2000): *Web Work: Information Seeking and Knowledge Work on the World Wide Web*. Dordrecht, Netherlands: Kluwer Academic Publishers.

Collins, H.M. (2001): Tacit knowledge, trust, and the Q of sapphire. *Social Studies of Science*, 31(1), 71–85.

Cook, S.D.N. & Brown, J.S. (1999): Bridging epistemologies: The generative dance between organizational knowledge and organizational knowing. *Organization Science*, 10 (4), 381-400.

Edwards, K. & Ladeby, K. (2005): Framework for Assessing Configuration Readiness. *Proceedings of the 3rd Interdisciplinary World Congress on Mass Customization and Personalization* (MCPC2005), Hong Kong, Sept. 18-21.

Forza, C. & Salvador, F. (2002): Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems. *International Journal of Production Economics*, 76, 87-98.

Forza, C., Trentin, A., & Salvador, F. (2005): Product Information Management for Mass Customization: the Case of Kitting. *Proceedings of the 3rd Interdisciplinary World Congress on Mass Customization and Personalization* (MCPC2005), Hong Kong, Sept. 18-21.

Frutos, J.D., Santos E.R. & Borenstein, D. (2004): Decision support system for product configuration in mass customization environments. *Concurrent Engineering Research and Applications* 12 (2), 131-144.

Gettier, E. (1963): Is Justified True Belief Knowledge? *Analysis*, 23, 121-123.

Gourlay, S.N. (2004): Knowing as semiosis: steps towards a reconceptualization of 'tacit knowledge'. In Tsoukas, H. & Mylonopoulos, N. (Ed.), *Organizations as Knowledge Systems* (pp 86–105). London: Palgrave Macmillan.

Gourlay, S.N. (2006): Towards conceptual clarity concerning 'tacit knowledge': a review of empirical studies. *Knowledge Management Research and Practice*, 4(1), 60-69.

Grant, R.M. (1996): Toward a knowledge-based theory of the firm. *Strategic Management Journal*, 17, 109-122.

Gustavsson, B. (2001): Vidensfilosofi [Philosophy of Knowledge]. (Hansen, I.B. Trans.). Aarhus, Denmark: Klim. (Original work published in 2000).

Hansen, B., Riis, J. & Hvam, L. (2003): Specification process reengineering: concepts and experiences from Danish industry. *Proceedings of the 10th ISPE International Conference on Concurrent Engineering: Research and Applications*, Madeira, Portugal, July 26-30.

Hansen, B.L. (2003): *Development of Industrial Variant Specification Systems*. PhD thesis. Department of Manufacturing Engineering and Management, Technical University of Denmark.

Heiskala, M., Paloheimo, K.-S. & Tiihonen, J. (2005): Mass Customisation of Services: Benefits And Challenges of Configurable Services. *Frontiers of e-Business Research (FeBR) 2005*, conference proceedings of eBRF, Tampere, Finland, Sept. 26.-28.

Hitt, M. A., Ireland, R. D. & Lee, H. (2000): Technological learning, knowledge management, firm growth and performance: an introductory essay. *Journal of Engineering and Technology Management*, 17, 231-246.

Hvam, L. (2004): A Multi-perspective approach for the design of Product Configuration Systems - An evaluation of industry applications. *Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems*, Lyngby, Denmark, June 28-29.

Hvam, L. & Malis, M. (2001): A Knowledge Based Documentation Tool for Configuration Projects. *Proceedings of World Congress on Mass Customization and Personalization*, Hong Kong, Oct. 1-2.

Hvam, L., Mortensen, N.H. & Riis, J. (2006): *Produktkonfigurering* [Product Configuration]. Copenhagen, Denmark: Nyt Teknisk Forlag.

Johnson, P. & Cassel, C. (2001): Epistemology and work psychology: New agendas. *Journal of Occupational and Organizational Psychology*, 74, 125–143.

Møldrup, M. & Møller, N. (2004): Development and implementation of product configuration systems – a change management perspective. *Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems*, Lyngby, Denmark, June 28-29.

Newell, S., Robertson, M., Scarbrough, H. & Swan, J. (2002): *Managing knowledge work*. Basingstoke: Palgrave Macmillan.

Nonaka, I. & Takeuchi, H. (1995): *The Knowledge-Creating Company*. New York: Oxford University Press.

Nonaka, I., Toyama, R. & Konno, N. (2000): SECI, ba and leadership: a unified model of dynamic knowledge creation. *Long Range Planning*, 33(1), 5-34.

Pedersen, J.L. & Edwards, K. (2004): Product Configuration Systems and Productivity. *Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems*, Lyngby, Denmark, June 28-29.

Polanyi, M. (1962): *Personal Knowledge.* Chicago: The University of Chicago Press. (Corrected edition, original from 1958).

Polanyi, M. (1966): *The Tacit Dimension.* Reprint, Gloucester, Mass.: Peter Smith, 1983.

Polanyi, M. (1969): Knowing and Being. In Grene, M. (Ed.), *Knowing and Being*. Chicago: University Of Chicago Press.

Riis, J. (2003): *Fremgangsmåde for opbygning, implementering og vedligeholdelse af produktmodeller - med fokus på konfigureringssystemer* [Procedure for building, implementing and maintaining product models - with focus on configuration systems]. PhD thesis. Department of Manufacturing Engineering and Management, Technical University of Denmark.

Ryle, G. (1949): *The concept of the mind*. Reprint, Chicago: University of Chicago Press, 1984.

Sabin, D. & Weigel, R. (1998): Product Configuration Frameworks - A survey. *IEEE Intelligent Systems & Their Applications*, 13(4), 42-49.

Schwarze, S. & Schönsleben, P. (1996): Recent Developments in Configuration of Multiple-Variant Products: Application Orientation and Vagueness in Customer Requirements. *Proceedings of the APMS'96 International Conference on Advances in Production Management Systems*, Kyoto, Japan, Nov. 4-6.

Skjevdal, R. & Idsoe, E.A. (2005): The Competitive Impact of Product Configurators in Mass Tailoring and Mass Customization Companies. *Proceedings of the 3rd Interdisciplinary World Congress on Mass Customization and Personalization* (MCPC2005), Hong Kong, Sept. 18-21.

Spender, J.-C. (1996): Competitive advantage from tacit knowledge? Unpacking the concept and its strategic implication. In Moingeon, B. & Edmondson, A. (Ed.), *Organizational Learning and Competitive Advantage* (pp. 56-73). London: Sage Publications.

Spender, J.-C. (1998): Pluralist epistemology and the Knowledge-based theory of the firm. *Organization*, 5(2), 233-256.

Stenmark, D. (2002): Information vs. Knowledge: The Role of intranets in Knowledge Management. *Proceedings of HICSS-35*, Hawaii, Jan. 7-10.

Tiihonen J., Lehtonen T., Soininen T., Pulkkinen A., Sulonen R. & Riitahuhta A. (1998): Modeling Configurable Product Families. *Proceedings of 4th WDK Workshop on Product Structuring*, Delft University of Technology, the Netherlands, Oct. 22-23.

Tsoukas, H. (2003): Do we really understand tacit knowledge? In *The Blackwell Handbook of Organizational Learning and Knowledge Management* (Easterby-Smith, M. & Lyles, M.A., Ed.), 410–427. Oxford: Blackwell Publishing Ltd.

Wiig, K. M. (1993): Knowledge Management Foundations: Thinking About Thinking – How People and Organizations Create, Represent, and Use Knowledge. Arlington, Texas: Schema Press.

Paper A2

A classification of the information that domain experts do and do not provide in configuration projects

Haug, A. and Hvam, L. (2007): "A classification of the information that domain experts do and do not provide in configuration projects", in Proceedings of the 12th annual international conference on Industrial Engineering Theory, Applications & Practice, Cancun, Mexico, Nov. 4-7, 2007.

A classification of the information that domain experts do and do not provide in configuration projects

Anders Haug and Lars Hvam

Abstract: The use of product configurators has in many cases shown to be a successful way of improving business processes for industrial companies by providing benefits such as shorter lead times, the use of fewer resources for the creation of product specifications, and improved products. When developing a product configurator, one of the biggest challenges is to elicit the domain knowledge that is to constitute the knowledge base of the product configurator. Many researchers have claimed that the tacit knowledge of domain experts is a main problem. However, this kind of explanation has recently been criticised by the author of this paper for giving a wrong picture of reality. In order to offer a more useful framework for analysing configuration projects this paper proposes a new classification of the types of information that domain experts do and do not deliver to knowledge engineers.

Keywords: Product configuration, Design automation, Knowledge engineering, Acquisition of engineering knowledge

1. INTRODUCTION

A product configurator is a software system that supports the specification of customized products by allowing the users to define products, while restricting how different components and properties may be combined. For several industrial engineering companies the use of product configurators has produced a range of benefits, such as: shorter lead times, improved quality of product specifications, preservation of knowledge, use of fewer resources for specifying products, optimized products, less routine work, improved certainty of delivery, and less time needed for training new employees (e.g. Ardissono et al., 2003; Hvam, 2004; Hvam et al., 2007; Forza and Salvador, 2002; Forza and Salvador, 2007).

When creating a product configurator to automate engineering work, one of the biggest challenges is to delimit, structure, and represent the product knowledge that forms the basis for the knowledge base of the product configurator (Sabin and Weigel, 1998; Hvam et al, 2007; Forza and Salvador, 2007; Hvam et al., 2006; Edwards et al., 2005). A major challenge is that much of the relevant knowledge is often not accessible in a well-structured form.

Much configuration literature claims that it is the tacit knowledge of domain experts that makes the task of representing domain expert knowledge difficult in configuration projects (e.g. Møldrup and Møller, 2004; Pedersen and Edwards, 2004; Hansen, 2003; Edwards and Ladeby, 2005; Tiihonen et al., 1998). However, Haug and Hvam (2007) have argued that the use of the term *tacit knowledge* to explain why the creation of knowledge representations in configuration projects can be problematic is likely to give a wrong picture of this issue. The arguments presented by Haug and Hvam (2007) are resumed in this paper, and a new way of classifying the knowledge/information of the domain experts in a product configuration context is proposed.

2. TACIT KNOWLEDGE IN A CONFIGURATION CONTEXT

2.1 Tacit knowledge

In literature from many different fields of research, knowledge is often divided into tacit and explicit knowledge. Explicit knowledge is normally defined as knowledge that is codified and can readily be communicated to others. However, there is a debate whether pure explicit knowledge actually exists or not, as some claim that pure explicit knowledge would merely be information, since even the most explicit kind of knowledge is underlain by tacit knowledge (Stenmark, 2002, Tsoukas, 2003).

Throughout most of history, western epistemology has viewed knowledge as being explicit and focused on defining clauses for when something can be qualified as knowledge. A classical definition of knowledge is "justified true beliefs", which can be traced back to Plato's "Theaetetus" (c.400 BC) and Kant's "Critique of pure reason" (from 1781) (Bernecker and Dretske, 2000). This tripartite account of knowledge has since received criticism from different angles, but is still used by many researchers, however, often in less strict forms.

Towards the 1980s literature about practical knowledge began to have a significant impact on the development of society. In this context some of the most important contributions are from Ludwig Wittgenstein, Gilbert Ryle, and Michael Polanyi (Gustavsson, 2001). The book "The concept of the mind" by Gilbert Ryle from 1949 (Ryle, 1949) was to be a central basis for the discussion of practical knowledge (Gustavsson, 2001). The central term that Ryle introduces is *knowing how*, which is put in contrast to the traditional form of knowledge, namely *knowing that*. The book "Personal knowledge" from 1958 by Michael Polanyi (Polanyi, 1969) is one of the most import contributions on the topic of tacit knowledge and was later followed by other publications about tacit knowledge by this author. One of the most cited quotes from Polanyi is the one of "we can know more than we can tell" (1966). The quote is among other supported by the example of a persons face which can be recognized among a thousand, while we usually cannot tell how we recognize a face we know. Therefore, most of this kind of knowledge cannot be put into words. Polanyi possesses a dynamic view on knowledge, captured in the words: "Knowledge is an activity which would be better described as a process of knowing" (Polanyi, 1969).

Tacit knowledge has been a popular term in management studies since the mid 1990s, to a large extent because of Nonaka and Takeuchi's "The Knowledge-Creating Company" (Nonaka and Takeuchi, 1995) (Tsoukas, 2003). However, some researchers argue that in this kind of literature Polanyi has been misunderstood, and that Polanyi's ideas may not be particularly relevant for understanding knowledge in organisations. Gourlay (2004) argues that although Polanyi was responsible for the phrase *tacit knowledge*, his later work shows that he was concerned with a process, namely tacit knowing, and not a form of knowledge. Tsoukas (2003) argues that Nonaka and Takeuchi interpret the term *tacit knowledge* as "knowledge not yet articulated" in the sense that this is knowledge that awaits *translation* or *conversion* into explicit knowledge. Tsoukas further argues that although such an interpretation has been widely adopted in management studies, it is erroneous, since it ignores the essential ineffability of tacit knowledge and reduces it to what can be articulated. According to Tsoukas, tacit and explicit knowledge should be seen as two sides of the same coin, and not two ends of a continuum, in that even the most explicit kind of knowledge is underlain by tacit knowledge.

In literature the ambiguities concerning the meaning of tacit knowledge are many. Gourlay (2006) reviews empirical studies of tacit knowledge and points out three distinct ways in which the term *tacit knowledge* has been applied: (1) in cases where the knowledge in question could be articulated; (2) in cases where only feelings to tacit knowledge were claimed; and (3) in cases where there was evidence of action/behaviour of which the actors could not give account (excluding inhibitions of communication). For clarity of communication Gourlay proposes that the term *tacit knowledge* is only applied about the third category of empirical phenomena, and that the issue of whether or not tacit knowledge can be made explicit is reframed in terms of "whether functionally equivalent descriptions of behaviours can be made". Haug and Hvam (2007) advocate that this perception of tacit knowledge should also be applied within product configuration research to ensure that a common language is used and to avoid the dilution of the term. However, no matter if this proposition is followed or a broader definition is accepted, the use of the term *tacit knowledge* in product configuration literature is problematic, as argued in the next section.

2.2 Problems of tacit knowledge in configuration research

As mentioned in section 1, several researchers have indicated that the tacit knowledge of domain experts is the dominant factor that renders problematic the creation of product configurators. This literature rarely provides detailed examples or definitions of how the term should be perceived. Obviously, in itself it is problematic to apply such an ambiguous term without further defining it, since it is impossible to know what it refers to in the given context. But furthermore, no matter which of the available definitions is chosen, the use of the term is still problematic from at least two points of views (Haug and Hvam, 2007).

The first problem of using the distinction between tacit and explicit knowledge to explain why the representation of some product knowledge can be difficult is that if the knowledge of a company is divided into only tacit and explicit knowledge, then what is not explicit must be tacit. Since only known explicit knowledge can be categorised as being explicit, there is a danger that a lot of different things are put into the category of tacit knowledge, including not yet articulated explicit knowledge, knowledge unknown to the observer, forgotten explicit knowledge, not yet created knowledge etc. To include all this kind of knowledge into the category of tacit knowledge contradicts the original definition of tacit knowledge and in principle dilutes the term to be "all knowledge that is not known explicit knowledge". On the other hand, if a narrow definition of tacit knowledge is applied, as advocated by the author of this paper, then something more than the tacit-explicit distinction is needed for describing the knowledge that emerges in the knowledge elicitation situation of a configuration project.

The second problem of using the distinction between tacit and explicit knowledge to explain why the representation of some product knowledge can be difficult is that tacit knowledge does not necessarily represent a problem, while explicit knowledge (or information) cannot by definition be considered unproblematic. For instance, in a product configuration context, where tacit knowledge of a domain expert is interesting, it must be presumed that the domain expert is at least capable of telling

what is done in specific situations, i.e. what the domain expert can infer. This means that the task of describing the choices made by a domain expert based on particular design requirements should not be a problem. On the other hand, it is incorrect to presume that explicit knowledge by definition is unproblematic, since in some cases explicit definitions can contradict each other, i.e. when different domain experts have different perceptions of what is *the right way of doing things*. Also in cases where the problem domain is complex and the knowledge engineer does not fully understand relevant knowledge, it can be difficult to create correct knowledge representations. Therefore, it can be problematic with too much focus on explicit knowledge as something which by definition is easy to deal with, and tacit knowledge as some sort of difficult knowledge in a product configuration context.

Having strong arguments for the fact that the distinction between tacit and explicit knowledge is somewhat unsuited for explaining why it can be difficult to create the needed knowledge representations in configuration projects, the task is then to find a better framework for understanding this issue.

3. A NEW CLASSIFICATION

3.1 Clarification of focus

In the early 1980s the development of expert systems (or knowledge based systems) was seen as a transfer process, where the assumption was that the required knowledge already existed and that it just had to be collected and implemented. Today there is an overall consensus that the process of building an expert system is better to be perceived as a modelling activity, i.e. the task is not perceived as being to copy domain expert knowledge, but to create a model that simulates the domain expert (Clancey, 1993; Studer et al., 1998; Speel et al., 2001). This corresponds with the perception of the author, for which reason this paper advocates for the application of terms such as *translation* and *synthesis* for describing the process of getting domain expert expertise into knowledge representations, instead of the term *transfer* which is commonly encountered in relevant literature.

Based on the configuration projects investigated by the Centre of Product Modelling at the Technical University of Denmark (e.g. described in Hvam 2004; Hvam, 2007), the process of getting from the situation where the relevant knowledge is possessed by domain experts to a situation where this knowledge is simulated by a product configurator can, somewhat simplified, be perceived as consisting of the five activities shown in figure 1 and subsequently explained. In some cases the knowledge engineers and programmers are the same persons, for which reason to some degree process 2 merges with process 4 and process 3 merges with process 5.

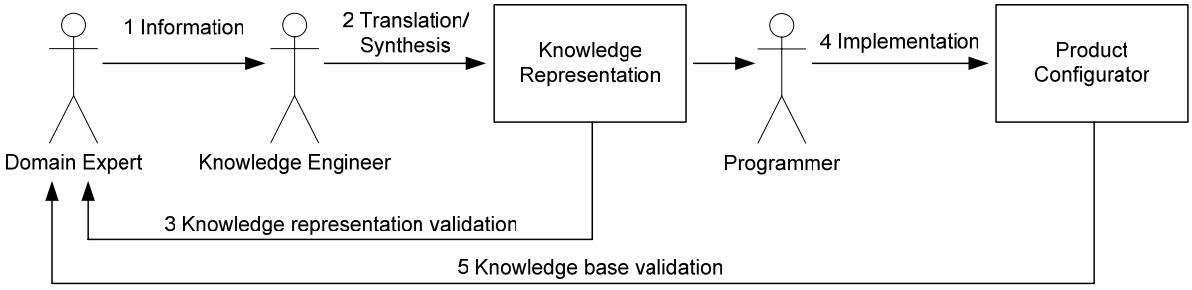


Figure 1. The creation of a product configurator

(1) The knowledge engineer questions a domain expert who delivers some information.
(2) Depending on the quality of the acquired information, the knowledge engineer translates the acquired information into the chosen knowledge representation languages or creates new knowledge based on the acquired information, sometimes in collaboration with domain experts. The

representation language for these kinds of models is often product variant masters or class diagrams (e.g. Hvam et al., 2007; Felfernig et al., 2000; Männistö et al., 2001).
(3) Domain experts evaluate if the created knowledge representations represent their domain in a satisfactory manner.
(4) The models created are implemented into the knowledge base of a product configurator. This step may include a translation of knowledge representations and often the creation of additional information which is required to make the product configurator operational.
(5) Domain experts evaluate the implemented knowledge by testing the product configurator.

In this paper the focus is on the first activity, but delimited to what is illustrated in figure 2, i.e. not including the part of the process concerning how a knowledge engineer interprets the information delivered by the domain expert.

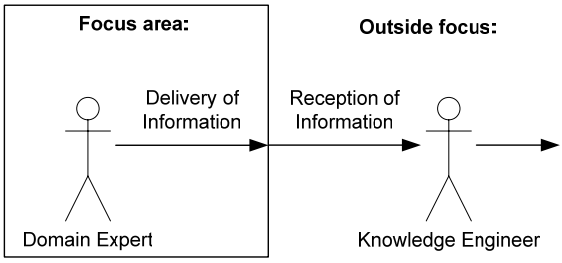


Figure 2. Focus of the paper

3.2 Collins' classification

Having argued that the use of the term *tacit knowledge* for describing knowledge acquisition difficulties can be misleading, or at best uninformative, there is a need for a classification which in a more detailed manner describes the knowledge that can cause problems in the communication between knowledge engineers and domain experts.

The studies of Collins (1974, 2001) are among the most thoroughly and well-documented studies of tacit knowledge. Collins (2001) offers a classification of knowledge that seems to provide a useful starting point in a product configuration context. Collins takes a basis in the fact that some scientists can do certain experiments while others cannot, and he points out that one of the reasons for this might be that the unsuccessful scientists lack tacit knowledge. In such a context, Collins defines tacit knowledge as "knowledge or abilities that can be passed on between scientists by personal contact but that cannot be, or has not been, set out or passed on in formulae, diagrams, or verbal descriptions and instructions for action". According to Collins at least five kinds of knowledge can be passed on by such personal contact:

(1) Concealed knowledge ("A" does not want to reveal his knowledge to others or the journals provide insufficient place to include relevant details)
(2) Mismatched salience (there is an infinite number of important variables where "A" does not realise that "B" needs to be told things in certain ways and "B" does not know what questions to ask)
(3) Ostensive knowledge (words, diagrams or photographs cannot convey information that can be understood by direct pointing, demonstrating or feeling)
(4) Unrecognised knowledge ("A" performs certain parts of an experiment without realising their importance and therefore does not tell "B" about these. "B" can pick up the same habits when working together with "A" without realising this)
(5) Uncognized/uncognizable knowledge (things such as speaking acceptably-formed phrases in a native language without knowing how it is done - such abilities can be passed on only through apprenticeship and unconscious emulation).

In a product configuration context it seems plausible that domain experts in some cases may conceal knowledge or not recognise that they possess the knowledge that a knowledge engineer may need. Therefore, of the mentioned types of knowledge in the classification by Collins, *concealed* and *unrecognised* knowledge seem to be relevant in the described form. On the other hand, *mismatched salience* does not qualify as a type of knowledge, but rather as something that may lead to *irrelevant* or *unrecognised* information from a domain expert. Both *ostensive knowledge* and *uncognizable knowledge* belong to a category of *inarticulable knowledge*. The line between the two kinds of knowledge can be difficult to define, and, therefore, it may be better to use the term inarticulable knowledge about both kinds of knowledge. Besides the mentioned kinds of knowledge, other types of problematic knowledge or information can be added: non-possessed knowledge, incorrect information, and contradicting information.

The preceding discussion leads to the classification presented in figure 3, which is further explained in the following. It should be noticed that the term *information* is used consistently instead of *knowledge*. This is done based on the assumption that all knowledge has a tacit dimension, for which reason what domain experts articulate is merely information until it is interpreted by a receiver. However, researchers with other views on knowledge can also apply the classification, but in such a case some of the information types may instead be perceived as knowledge types.

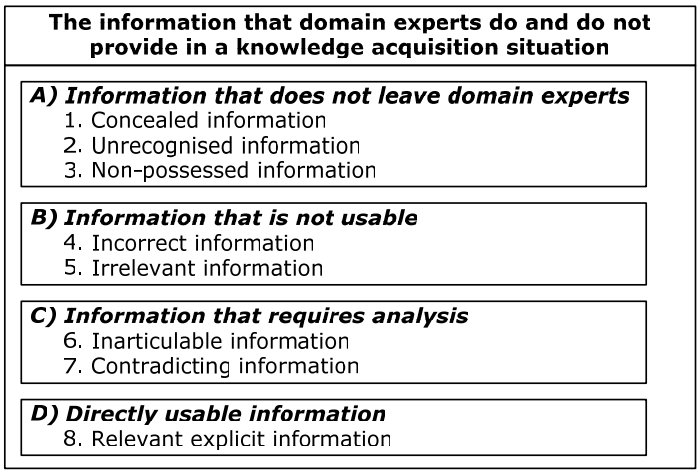


Figure 3. The information provided and not provided by domain experts during knowledge acquisition

A1. Concealed information: In the cases where the purpose of implementing a product configurator is for the product configurator to do tasks that were previously carried out by human experts, it seems plausible that some domain experts may hide information from the knowledge engineer, since the product configurator could mean loss of expertise for a domain expert, which, in the worst case, could lead to the redundancy of the domain expert.

A2. Unrecognised information: Unrecognised information could occur in a situation where a domain expert does not provide complete descriptions of the relevant domain, because the domain expert fails to recognise the relevance of some expertise that he/she possesses.

A3. Non-possessed information: A situation that may occur in a product configuration project is that a knowledge engineer thinks that a particular domain expert possesses the required expertise, which, however, the domain expert does not. The information that the knowledge engineer is looking for may even not exist and needs to be constructed to form the basis for the creation of a product configurator.

B4. Incorrect information: The domain expert may pass on incorrect information due to several reasons. The reason could be that the domain expert, as in the case of *concealed information*, does not want to share his/her expertise and therefore provides incorrect information. The reason for giving

incorrect information could also be pride, in that the domain expert does not want to admit he/she does not know the answer to the questions form the knowledge engineer and therefore provides guesses that may be incorrect.

B5. Irrelevant information: As mentioned, irrelevant information from a domain expert could be the effect of mismatched salience. The domain expert may be focussing on different aspects than the knowledge engineer, which is why the domain expert provides irrelevant information. Also the knowledge engineer may lack a proper understanding of the domain investigated and ask the domain experts the wrong questions. This could obviously lead to that the knowledge engineer is retrieving irrelevant information.

C6. Inarticulable information: A domain expert may posses some knowledge that he/she is incapable of articulating. This is typically *know-how* that allows a domain expert to perform certain actions and corresponds to the definition of tacit knowledge, which this paper suggests should be applied in product configuration research. However, not being able to articulate the required information does not mean that this information cannot be applied by the knowledge engineer. Observing the domain expert performing the relevant actions, which the domain expert cannot explain how he/she does, could be a way of creating the explicit descriptions that simulates what the domain expert is doing.

C*7. Contradicting information:* Contradicting information can occur without any of the information being decidedly incorrect. This, for instance, in the case where one domain expert claims that one solution is the best for a given problem while another domain expert claims that another solution is the best for the given problem. The two domain experts may both be right, depending on which aspects the focus is on. The first domain expert may be focussing on the solution from an economical perspective while the second domain expert may focus on aesthetics.

D8. Direct usable information: Directly usable information is information that is relevant and expressed in an explicit fashion. However, this kind of information may still need translation to a knowledge representation language before being represented in a conceptual model or implemented into a product configurator.

4. EMPIRICAL INVESTIGATIONS

To investigate the usefulness of the proposed classification of the information that a domain expert does and does not pass on to a knowledge engineer, four configuration projects were investigated. The investigation was carried out in the form of interviews with knowledge engineers from the configuration projects.

4.1 Cases

The four configuration projects that were investigated were carried out in ETO (Engineer-To-Order) companies. This type of project was chosen to ensure a certain amount of complexity of the information that was to be extracted from domain experts. Another selection criterion was that the knowledge engineer had to have been participating in a high number of knowledge acquisition sessions. The interviews with the knowledge engineers were given on the condition of anonymity, which is why the four companies are named A through D. In table 1 an overall description of the cases studied is shown.

Table 1. Overview of studied cases

	Period for knowledge acquisition	Knowledge engineer employment terms	Domain experts interviewed *	Domain expert characteristics
Case A	12 months	External	4 primary, ~ 10 in total	Mainly young skilled workers
Case B	5 years **	Employee	~ 40	Mainly experienced engineers

Case C	5 months	External	3	Experienced engineers
Case D	18 months	Employee	5	Experienced engineers

* Other domain experts could supply information indirectly

** Primarily focusing on the first years of project

In all the cases the product configurator had been taken into use, and the task of representing the relevant domain knowledge was carried out between recently and 6 years ago. This lapse of time, obviously, could have an effect on what the knowledge engineers were able to remember. On the other hand, this downside was considered to be less significant than dealing with less extensive or completed projects, where some of the defined types of information could not have been encountered yet because of the stage of the project or because of the lack of complexity of the product knowledge involved.

4.2 Method

The four projects were investigated by interviewing a knowledge engineer from each of the projects. The interviews were designed as semi-structured interviews. For each of the defined information types the interviewees were firstly asked if they had encountered the specific type of information as a knowledge engineer in the specific product configuration project. If answering "yes", they were asked to provide examples of situations where this had occurred, and to give any additional comments they might have. In this part of the interview new questions were created in the situation in order to get as much information as possible from the interviewees. After repeating this sequence for each of the eight types of information, the interviewees were asked to estimate how frequently the different types of information occurred. Obviously, it could be almost impossible for the interviewed knowledge engineers to account for how often the defined types of information occurred in the measure of percentage of the time or percentage of sessions. Therefore, the knowledge engineers were asked to use the scale of: never, seldom, occasionally, and often. Obviously, the interviewed knowledge engineers may have slightly different perceptions of what frequency these words indicate. However, the main point of the investigations was not to find out how often the defined types of information occur, but rather if they occur and in what type of situations.

After these questions, the interviewees were asked if they could name other types of relevant information in the given context, besides the included in the proposed classification. The interviews were taped and notes were taken during the interviews. In case B and C the knowledge engineers were contacted after the interview for clarification of some uncertainties.

4.3 Results

In table 2 the results of the interviews are shown and subsequently further explained.

Table 2. Occurrence of the information types

Knowledge type	**Case A**	**Case B**	**Case C**	**Case D**
A1. Concealed information	Seldom	Never/Seldom	Never	Never/Seldom
A2. Unrecognised information	Occasionally	Occasionally	Seldom	Seldom
A3. Non-possessed information	Often	Occasionally	Often	Occasionally
B4. Incorrect information	Seldom	Seldom	Seldom	Occasionally
B5. Irrelevant information	Often	Often	Often	Often
C6. Inarticulable information	Seldom	Seldom	Never	Never
C7. Contradicting information	Often	Often	Seldom	Occasionally
D8. Relevant explicit information	Most often	Most often	Most often	Most often

A1. Concealed information: Generally, the judgement of the interviewed knowledge engineers was that domain experts concealing information had not been a great problem. Only in case A the interviewed knowledge engineer strongly suspected that some domain experts in rare cases may have withheld information. This can be explained by the fact that some of the domain experts in case A were young career-oriented persons aspiring to be sales persons, and that the product configurator to some extent could make their sales expertise superfluous. On the other hand, this was not the situation in the other cases, where the domain experts according to the interviewed knowledge engineers mostly did not see the product configurator as a competitor, but more as a helpful tool. However, in case B there were minor problems with some domain experts that were reluctant to waste time on the configuration projects, as they prioritised other tasks.

A2. Unrecognised information: According to the interviews, unrecognised information was more frequently encountered in case A and B than in case C and D. This can be explained by the fact that the products of case A and B were more extensive and complex than in case C and D. Another explanation is that in case C and D a few domain experts were given the responsibility of collecting information from other domain experts and deliver this to the knowledge engineer, for which reason the knowledge engineer would not be involved in many of the situations where unrecognised information may occur. While unrecognised information occurred occasionally in both in case A and B, according to the interviewees, at some point this unrecognised expertise emerged from activities such as talking to other domain experts, rephrasing questions, and showing domain experts unfinished conceptual models.

A3. Non-possessed information: Domain experts who did not posses the required information occurred occasionally or often in the investigated cases. However, according to the interviewed knowledge engineers in case A, B and C, this did not represent a big problem, since in the situations where the domain experts did not posses the required information, at least they most often knew who else did. On the other hand, in case D there were some problems because some of the required information had not yet been created, since company D were redefining their products at the same time as the product configurator was defined. The knowledge engineer therefore had to make the domain experts invent new generic rules that were required in order to build a product configurator.

B4. Incorrect information: In all the cases the knowledge engineers claimed that some domain experts at some point delivered incorrect information. However, all the interviewed knowledge engineers agreed that this was probably not intentionally done, but because of incorrect guesses. When providing guesses, the domain experts most often gave a reference to another domain expert for checking the information or agreed to investigate the issue closer themselves. According to the interviewees it seems that incorrect information was a greater problem in case D. This can be explained by the fact that, as mentioned, in case D many of the product definitions were created in parallel with the conceptual models. This meant that the defining of the products included some trail and error for which reason the defined product specifications occasionally resulted in poor solutions and had to be redefined, just as the conceptual models had to be.

B5. Irrelevant information: According to all the interviewed knowledge engineers, domain experts often provided irrelevant information. As pointed out by all the interviewed knowledge engineers, at the beginning of a project it can be very difficult for an knowledge engineer to overview what may later be relevant, for which reason much information that later turns out to be irrelevant is included in the early versions of the conceptual models. Furthermore, the knowledge engineer of case A claimed that it is typical that domain experts like to talk about what they know and sometimes can be a bit reluctant to admit that they do not know the answer to a question.

C6. Inarticulable information: Judging from the investigated cases, inarticulable information is not a significant problem in configuration projects. Only in case A and B the knowledge engineers had the suspicion that the domain experts possessed inarticulable information or knowledge that was required for the creation of the conceptual models. In case C and D the domain experts were asked to prepare explicit information to each modelling session, which obviously to a great extent could limit such a problem. Case B provided a good example of inarticulable knowledge. In this case domain experts, by using special software, were able to make a simulation of the behaviour of a specified, but not yet created, product, although unable to fully explain how they did this.

C7. Contradicting information: According to the interviewed knowledge engineers in case A and B, contradicting information was a great problem, while it was less problematic in case C and D. The knowledge engineer of case C explained the low frequency of this problem by the homogeneousness of the small team of interviewed domain experts, while the knowledge engineer of case D explained that the reason why this did not occur more often was because the products were relatively simple to describe, and that relatively few domain experts were interviewed. On the other hand, the relatively large amount of contradicting information in the cases A and B was explained by the high number of interviewed domain experts. One of the major disagreements in case A concerned the question of which components should be included in the product configurator and which ones should be left out. In case B one of the disagreements concerned the information included in the text-outputs from the product configurator, an issue seen very differently from the point of the sales persons, product specialists, managers etc.

D8. Direct usable information: Fortunately, in all the cases, the knowledge engineers found that the main amount of information provided by the domain experts were explicit and relevant.

Applicability of classification: As mentioned, the knowledge engineers of the four cases were asked if they thought that the eight defined types of information are relevant, and if they could think of other kinds of information. In all the cases the knowledge engineers stated that it was very plausible that the 8 kinds of information occurred in product configuration projects, and that all the eight types of information, therefore, are relevant to include in the classification. On the question of whether they could think of other kinds of information or not, only one of the interviewed knowledge engineers attempted to do so, by suggesting *information not understood by a knowledge engineer*. However, after discussing this, the interviewed knowledge engineer agreed that this should not be considered a type of information in the current classification, since the receiving of information by the knowledge engineer is outside the scope of the classification (as shown in figure 2).

4.4 Discussion

The studies carried out provided some interesting answers. All the interviewed knowledge engineers found that the proposed classification is useful and covers all types of relevant information in the given context. It is interesting to observe that according to the knowledge engineers inarticulable information (or tacit knowledge) played such an insignificant role despite the seemingly claims of much product configuration research. It seems that the real challenges related to product configuration projects are to create not yet defined information and making domain experts agreeing on what information to use, i.e. dealing with contradicting information. Also deciding what is relevant to include in a model can be a challenge for knowledge engineers, particularly in the early phases of a project (avoiding irrelevant information).

5. CONCLUSION

This paper took a basis in an earlier paper by the author in which the author criticizes much product configuration literature for using the term *tacit knowledge* to explain why knowledge elicitation can be difficult in configuration projects (Haug and Hvam, 2007). This criticism concerns that tacit knowledge itself (according to the definition by the source of this term, i.e. Polanyi) is not likely to play a significant role in product configuration projects, while other types of knowledge or information are. Based on (Haug and Hvam, 2007), it was argued that a strict meaning of the term *tacit knowledge* is required for the sake of communication and for not diluting the term *tacit knowledge* to mean almost anything. Therefore, in a product configuration context tacit knowledge should be used only about inarticulable knowledge. This, however, leaves a void that needs to be filled by new classifications.

Inspired by the five types of knowledge defined by Collis (2001), a classification including eight types of information that domain experts do or do not provide to a knowledge engineer in a knowledge elicitation situations was proposed. The types are: concealed information, unrecognised information, non-possessed information, incorrect information, irrelevant information, inarticulable information,

contradicting information, and relevant explicit information. The first three types can be classified as information that does not leave the domain expert, the next two types as information that is not usable, the next two as information that requires analysis, and the last one as directly usable information.

To investigate the relevance of the defined types of information, investigations of four product configuration projects were carried out by interviewing the knowledge engineers of the respective cases. Interestingly, in the four cases the investigations showed that according to the knowledge engineers inarticulable information (or knowledge) played an insignificant role. Instead, the investigations showed that what seems to be the real challenges in product configuration projects are: how to create not yet defined information (non-possessed information), how to make domain experts agree on what information to use (neutralise contradicting information), and decide what is relevant to include in a model (avoiding irrelevant information). Finally, all the interviewed knowledge engineers agreed that the proposed classification is useful and covers the types of information that domain experts do and do not provide in a configuration project.

While much configuration literature almost solely apply the term *tacit knowledge* to describe knowledge that represents difficulties in configuration projects, this paper presented seven kinds of information that could represent problems in a knowledge acquisition situation. Therefore, compared to existing research this paper has provided a much more nuanced basis for future configuration research. Besides being useful in a product configuration context, the classification may also be applicable in other lines of research that deals with situations where information is elicited from domain experts.

6. REFERENCES

1. Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Petrone, G., Schäfer, R., and Zanker, M. (2003). A Framework for the Development of Personalized, Distributed Web-Based Configuration Systems. AI Magazine, 24(3): 93-108.
2. Bernecker, S. and Dretske, F. (2000). In Knowledge: Readings in Contemporary Epistemology (Ed. S. Bernecker and F. Dretske). Oxford University Press, New York, pp. 3-6.
3. Clancey, W. J. (1993). The Knowledge Level Reinterpreted: Modelling Socio-Technical Systems. International Journal of Intelligent Systems, 8: 33-49.
4. Collins, H.M. (1974). The TEA set: tacit knowledge and scientific networks. Science Studies, 4: 165–186.
5. Collins, H.M. (2001). Tacit knowledge, trust, and the Q of sapphire. Social studies of science, 31(1): 71–85.
6. Edwards, K., Hvam, L., Pedersen, J.L., Møldrup, M., and Møller, N. (2005). Udvikling og implementering af konfigureringssystemer: Økonomi, Teknologi og Organisation [Development and implementation of configuration systems: Economy, Technology and Organisation]. Department of Manufacturing Engineering and Management, Technical University of Denmark.
7. Edwards, K. and Ladeby, K. (2005). Framework for Assessing Configuration Readiness. In Proceedings of the 3rd Interdisciplinary World Congress on Mass Customization and Personalization (MCPC2005), The Hong Kong University of Science and Technology.
8. Felfernig, A., Friedrich, G., and Jannach, D. (2000). UML as domain specific language for the construction of knowledge based configurations systems. International Journal on Software Engineering and Knowledge Engineering, 10(4): 449-470.
9. Forza, C. and Salvador, F. (2002). Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems. International Journal of Production Economics, 76: 87-98.
10. Forza, C. and Salvador, F. (2007). Product Information Management for Mass Customization. Palgrave MacMillan, New York.

11. Gourlay, S.N. (2004). Knowing as semiosis: steps towards a reconceptualization of 'tacit knowledge'. In Organizations as Knowledge Systems (Ed. H. Tsoukas and N. Mylonopoulos). Palgrave Macmillan, London, pp 86-105.
12. Gourlay, S. (2006). Towards conceptual clarity for 'tacit knowledge': a review of empirical studies. Knowledge Management Research & Practice, 4(1): 60-69.
13. Gustavsson, B. (2001). Vidensfilosofi [Philosophy of Knowledge] (Trans. I.B. Hansen, original work published in 2000). Klim, Aarhus, Denmark.
14. Hansen, B.L. (2003). Development of Industrial Variant Specification Systems (PhD thesis). Department of Manufacturing Engineering and Management, Technical University of Denmark.
15. Haug, A. and Hvam, L. (2007). Tacit knowledge in configuration projects. In Innovative Processes and Products for Mass Customization (Proceedings of the Joint Conference IMCM'07 & PETO'07), GITO-Verlag, Berlin.
16. Hvam, L. (2004). A Multi-perspective approach for the design of Product Configuration Systems - An evaluation of industry applications. In Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems, Technical University of Denmark, pp. 13-25.
17. Hvam, L., Mortensen, N.H., and Riis, J. (2007). Produktkonfigurering [Product Configuration]. Nyt Teknisk Forlag, Copenhagen, Denmark.
18. Hvam, L., Pape, S., and Nielsen, M.K. (2006). Improving the quotation process with product configuration. Computers in Industry, 57: 607–621.
19. Møldrup, M. and Møller, N. (2004). Development and implementation of product configuration systems – a change management perspective. in Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems, Technical University of Denmark, pp. 45-57.
20. Männistö, T., Soininen, T., and Sulonen, R. (2001). Modelling Configurable Products and Software Product Families. In Proceedings of the 7th International Joint Conference on Artificial Intelligence, Seattle, Washington, pp. 64-70.
21. Nonaka, I. and Takeuchi, H. (1995). The Knowledge-Creating Company. Oxford University Press, New York.
22. Pedersen, J.L. and Edwards, K. (2004). Product Configuration Systems and Productivity. In Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems, Technical University of Denmark, pp. 165-176.
23. Polanyi, M. (1966). The Tacit Dimension (Reprint 1983). Peter Smith, Gloucester, Mass.
24. Polanyi, M. (1969). Knowing and Being. Essays by Michael Polanyi (Ed. M. Grene). University Of Chicago Press, Chicago.
25. Ryle, G. (1949). The concept of the mind (Reprint 1984). University of Chicago Press, Chicago.
26. Sabin, D. and Weigel, R. (1998). Product Configuration Frameworks - A survey. IEEE Intelligent Systems & Their Applications, 13(4): 42-49.
27. Speel, P.H., Schreiber, A.T., Joolingen, W., Van Heijst, G., and Beijer, G.J. (2001). Conceptual Models for Knowledge-Based Systems. In Encyclopedia of Computer Science and Technology. Marcel Dekker Inc., New York.
28. Stenmark, D. (2002). Information vs. Knowledge: The Role of intranets in Knowledge Management. In Proceedings of HICSS-35, IEEE Press.
29. Studer, R., Benjamins, V.R, and Fensel, D. (1998). Knowledge Engineering: Principles and Methods. Data & Knowledge Engineering, 25: 161-197.
30. Tiihonen, J., Lehtonen, T., Soininen, T., Pulkkinen, A., Sulonen, R., and Riitahuhta, A. (1998). Modeling Configurable Product Families. In Proceedings of 4th WDK Workshop on Product Structuring, Delft University of Technology, the Netherlands.

31. Tsoukas, H. (2003). Do we really understand tacit knowledge? In The Blackwell Handbook of Organizational Learning and Knowledge Management (Ed. M. Easterby-Smith, M.A. Lyles). Blackwell Publishing Ltd., Oxford, pp. 410-427.

Paper B1

Product analysis as a basis for building product configuration systems

Haug, A. and Hvam, L. (2005): "Product analysis as a basis for building product configuration systems", in Proceedings from the 3rd Interdisciplinary World Congress on Mass Customization and Personalization (MCPC'05), Hong Kong, Sept. 18-21, 2005.

Product analysis as a basis for building product configuration systems

Anders Haug and Lars Hvam

Abstract

In this paper a notation technique for elicitation of product knowledge as a basis for building the knowledge base of a product configuration system (PCS) is proposed. The proposed notation technique is based on the so-called product family master plan (PFMP). The PFMP is part of a seven phase procedure for building product configuration systems (PBPCS) that has been applied in a number of projects in Danish industry during the last ten years. There are several arguments for proposing a new notation technique for elicitation of product knowledge. Firstly, there have in some cases been requests for an increased formalism of the PFMP technique to secure that relevant aspects are included and to avoid inappropriate use. Secondly, there seems to be a need for instructions in how to model a product through life phase systems, using the structural principles from the PFMP. The third argument is the ambition of creating an IT-based modelling and documentation system, which includes a user-friendly notation technique for elicitation of product knowledge that possesses a higher degree of formalism than the existing definitions of the PFMP.

As a basis for proposing the new notation technique, studies of the use of the PFMP in company projects were carried out.

Keywords: Product configuration, Product modelling, Knowledge representation

1 INTRODUCTION

The use of product configuration systems (PCS) is a technology that supports the manufacturing paradigm of mass customisation. A PCS is a product-oriented expert system, capable of carrying out knowledge-related tasks that are normally done by human experts. One of the big challenges when developing PCS is to formulate the product knowledge in an explicit and formalised manner, both as a basis for building the PCS and in later maintenance of the knowledge base as product families evolve. To support the development and maintenance of a PCS structured procedures can be applied. A procedure for building product configuration systems (PBPCS) was introduced in 1994 by Hvam [1] and has since undergone continuous developments at the Centre of Product Modelling at the Technical University of Denmark (CMP/DTU). The procedure (or part of the procedure) has been tested in projects at several companies, e.g. F. L. Smidth, American Power Conversion (APC), Aalborg industries, NEG-Micon, GEA-Niro and IBM-SMS [2]. The PBPCS is outlined in figure 1.

Phase	Description
1	**Process Analysis** Analysis of the existing specification process (AS-IS), statement of the functional requirements to the process. Design of the future specification process (TO BE). Overall definition of the product configuration system to support the process. **Tools:** IDEF0, flow charts, Activity Chain Model, key numbers, problem matrix, SWOT, list of functional describing characteristics and gap analysis.
2	**Product Analysis** Analysing products and eventually life cycle systems. Redesigning/ restructuring of products. Structuring and formalising knowledge about the products and related life cycle systems in a product variant master. **Tools:** List of features and product variant master.
3	**Object Oriented Analysis** Creation of object classes and structures. Description of object classes on CRC-cards. Definition of user interface. Other requirements to the IT solution. **Tools:** Use cases, screen layouts, class diagrams and CRC-cards.
4	**Object Oriented Design** Selection of configuration software. Defining and further developing the OOA-model for the selected configuration software. Requirements specification for the programming including user interface, integration to other IT-systems.
5	**Programming** Programming the system based on the model. Testing the configuration system
6	**Implementation** Implementation of the product configuration system in the organisation. Training users of the system, and further training of the people responsible for maintaining the product configuration system.
7	**Maintenance** Maintenance and further development of the product and product related models.

Figure 1: The procedure for building product configuration systems [3]

In spite of the waterfall model look on the representation of the PBPCS, the phases of the procedure are not intended to be performed sequentially, but rather in a concurrent and iterative way. Furthermore, the procedure should to some extend be regarded as an open development model. The procedure primarily aims at the use of standard configuration systems, e.g. the ones from SAP, Oracle and CinCom (descriptions of characteristics of several standards systems can be found at www.productmodels.org).

One of the strengths of the PBPCS is its completeness, as it provides support for other aspects of a PCS project than software development activities, such as organisational, economic and manufacturing aspects. Another more specific strength of the PBPCS is its use of techniques for elicitation of product knowledge. The procedure is based on the assumption that the model-managers (responsible for the product model, often industrial engineers) and domain experts (product and process experts) should be able to carry out the work of defining, and to some extend constructing, the

PCS without the need of support from IT-specialists. Preferring product expertise over IT-knowledge obviously demands that the applied techniques are relatively comprehensible for persons without a thorough knowledge of IT.

The procedure has separate phases with different knowledge representation techniques for respectively describing the products in the scope (phase 2) and for creating a more formal definition of the knowledge base along with specification of other system features of the PCS (phase 3-4). While the design phases are supported by UML, the preceding product analysis phase uses the so-called Product Family Master Plan (PFMP), which is described later in this paper.

The main argument for regarding the product analysis and PCS design as separate phases is that the elicited knowledge from domain experts may not have a form suitable for transferring directly to a PCS, especially in cases where the product documentation is produced by the domain experts themselves. Further, there might be a need for modularisation/standardisation of the product assortment before building the PCS, which often makes it necessary that the description of the product knowledge is formulated in a way the domain experts understand in order for them to be able to discuss these important decisions.

PFMP and class diagrams have not both been used in all the projects in which the PBPCS has been applied. Their application depends on aspects such as the modelling environment of the standard PCS (or programming language), complexity of the product, information modelling prerequisites of domain experts, etc.

In the textbook used for education in product configuration at DTU [4] the descriptions of how to apply the PFMP are formulated in a somewhat informal manner. For the experienced model-manager it might be adequate and even an advantage to have this freedom regarding the use of this technique. For the inexperienced model-manager these descriptive freedoms can often have the opposite effect such as doubts as how to apply the notation technique and inappropriate use. This has often been observed when students have used the technique in exercises or in real projects, and has lead to several requests for a more formal description of how to apply the notation technique. For modelling a product through its life phase systems the instructions of how to apply the PFMP are very limited and further it seems that a new notation principle is needed for modelling of this aspect in an efficient manner. Another important aspect is the long lasting ambition of CPM/DTU to create an IT-based modelling and documentation system, supporting the use of both the PFMP and class diagrams. So far, the basis has not been satisfactory for the initiation of this system development, where one of the major obstacles is the lack of formal definitions of how to use the PFMP technique.

In this paper a new notation formalism based on the PFMP is proposed. The introduction of this new notation technique aims to comply with all the mentioned interests. Besides presenting a diagram for the modelling of product structures, an extension of the technique for modelling product meetings with life phase systems is proposed. This paper further includes a discussion of the existing theoretical basis, concerning which are the views of a product that are relevant to model in a configuration context.

The remainder of this paper is structured as follows: In section 2, the theoretical foundations of product analysis according to the PBPCS is presented, followed by descriptions of the PFMP technique. After this, studies of the use of the PFMP in PCS projects are described in section 3. In section 4, a new notation technique for describing generic product models is presented. Finally, in section 5 conclusions are drawn and future research considered.

2 THE PRODUCT ANALYSIS PHASE

The way of perceiving products in the PBPCS is based on system theory, more specifically **theories of technical systems**. Theories of technical systems are descriptive theories for synthesis of human-made artefacts. According to Hansen and Andreasen [5] the comprehensive theory formulated by Hubka and Eder [6] has been recognised as a general theory of technical systems. Another big theoretical influence on the product understanding in the PBPCS is the **theory of domains**, introduced by Andreasen [7].

Hubka and Eder [8] define a transformation system as consisting of four interacting sub-systems: a technical process system, a technical system, an active environment and a human system. In a technical system the attributes of an operand are changed, as the operand goes through the technical process with an input state to an output state. Hubka and Eder suggest that the interior of a technical system can be described on the abstraction levels: functions structure, organ structure and component structure. The original theory of domains has four similar views on a technical system, which by Andreasen [7] are termed (translated from Danish) the domains of: process, function, organ and construction (later termed the **part domain**). These four views are illustrated by the so-called chromosome model, which elucidates the causality between processes, functions, organs and parts [9]. Later Mortensen [10] suggests that modelling of technical systems include both constitutive and behavioural models, where behaviour is further divided into intended (Soll) behaviour and resulting (Ist) behaviour from the realised constitutive model. In this view, functions are perceived as behaviour of organs and only three domains remain. The revised chromosome model is seen in figure 2. This model represents the most commonly used definition of the domains in the domain theory (e.g. [11]).

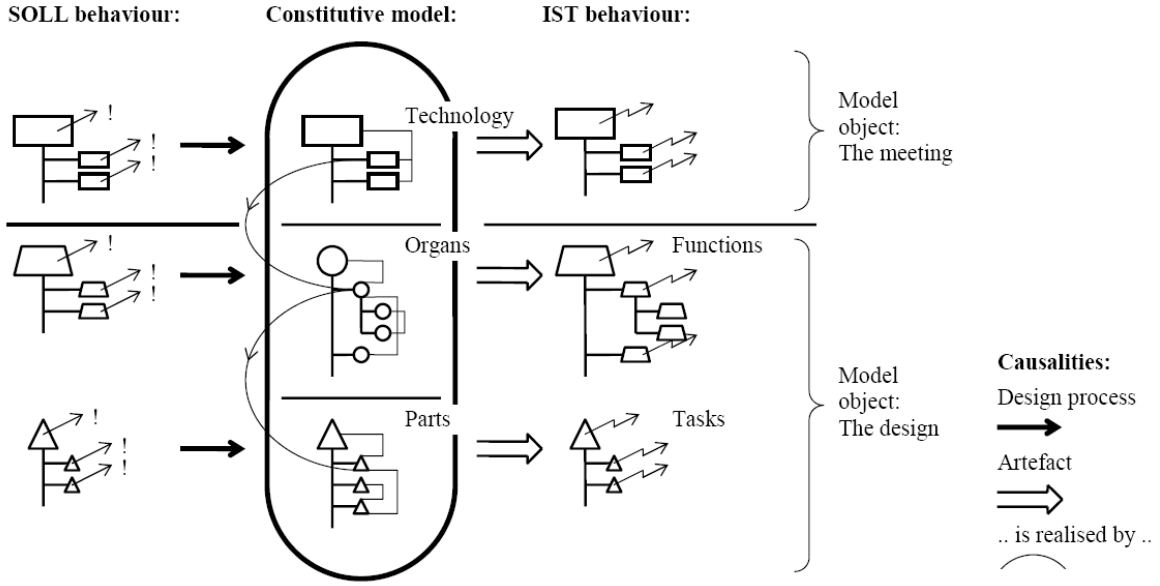


Figure 2: The revised chromosome product model [10]

Another influence on the view of product models in the PBPCS is **theory of dispositions** by Olesen [12]. The theory of dispositions focuses on the relation between a design and the life-phase activities of products. Olesen states that: 'by disposition we understand the part of a decision taken within one functional area that affects the type, content, efficiency or progress of activities within other functional areas'. The concept of dispositions implies that the designer controls aspects that exceed his working-situation and to a great extend includes working-situations of others. For instance, decisions concerning product structure affect the efficiency of the later assembly of a product. Olesen mentions several life phase systems that a product can meet during its life planning: fabrication, assembly, testing, transport, sales, installation, operation, service, scrapping, recycling and deposition. Olesen further introduces seven universal virtues to measure the performance of activities: cost, quality, flexibility, efficiency, lead-time, risk, and environmental effects. These virtues are claimed to be sufficient and complete classes of effects [12].

Besides the outlined theories in this paper, several other sub-divisions of product information exist, but the ones mentioned are the most important foundations of the PBPCS. This understanding is elucidated in the so-called framework for product models [1,13]. The framework for product models divides product information, which can be generic or instances, into two levels of sub-models as seen in figure 3.

Hubka and Eder [8] divide properties into two main sub-sections: external properties and internal properties. The term **external properties** is used about the properties of technical systems that are easily observed, either by human senses or assisting devices, such as measuring or sensing systems.

External properties are divided into eleven classes of properties: function, functionally determined, operational, ergonomic, aesthetic, distribution, delivery plus planning, law conformance, manufacturing, economic and liquidation. The **internal properties**, on the other hand, are so difficult to observe that it would require an expert to determine their existence and measure. This kind of properties includes e.g. strength, corrosion resistance and durability.

According to Hubka and Eder [8] the means of the designer for achieving all types of properties is **elementary design properties**. For higher levels of complexity of a technical system they consist of structure (comprising function, organ and component) and for the elementary technical system: form (shape), size (dimension), materials, surface (texture, quality), tolerances (dimensional, geometric) and method of manufacturing.

Property models:	Internal and external properties
	Functional properties
Product models:	Organ model
	Part model
Models of product relations to life phase systems:	Factory model
	Process model
	Assembling model
	Transport model
	Other life phase models

Figure 3: Framework for product models (Based on [13])

Hubka and Eder [8] define the external property **function** as being a superior property, as if the product does not include the right function it would be pointless to evaluate other properties. According to Andreasen [7] a function is defined as the ability of a machine to deliver a purposeful effect (e.g. light, heat or force).

Organs realises functions and are constituted by one or several parts. A specific part can also be a constituent of several organs and the part and organ structure are usually not identical [14]. Both organs and parts carry properties.

A **part** (from machine part) is defined as an elementary element produced by one material without assembly operations [7]. When designs are modelled as parts, the elements are parts or assemblies (complex parts) and their relations spatial [10].

2.1 The Product Family Master Plan

The most important (and often the only used) notation technique in the product analysis phase of the PBPCS is the Product Family Master Plan (PFMP). The technique is in some contexts referred to as a Product Variant Master, but in this paper we use the first mentioned term. The PFMP consists of two generic sections, part-of structure and kind-of structure. The part-of structure defines the parts that are in the model, while the kind-of structure displays different types of parts/modules/units that can be chosen. In figure 4 the notation formalism of the PFMP is shown.

The PFMP is normally drawn by using graphical software and then printed on a large piece of paper. The printout is discussed in sessions, including domain experts and model-managers, leading to a continual refinement of the PFMP. The PFMP is often combined with adapted CRC-cards [16] in which a card with further descriptions is produced for each part/module/unit.

Elicitation of product knowledge as a basis for building the knowledge base of a PCS can in many cases involve many domain experts, some of whom are only exposed to the knowledge representations in shorter periods of time. Therefore, high learnability of a notation technique would result in fewer resources spent on training of domain experts. Originally, the PBPCS [1] did not include the PFMP, and instead class diagrams were intended to be used for capturing the product knowledge of the

domain experts. The early projects in which the procedure was applied faced several problems in making domain experts use class diagrams, which led to the fact that the PFMP became incorporated into the procedure. The experience gained from the use of the PFMP has so far been positive [4], for which reason the technique is considered to be a central element of the PBPCS.

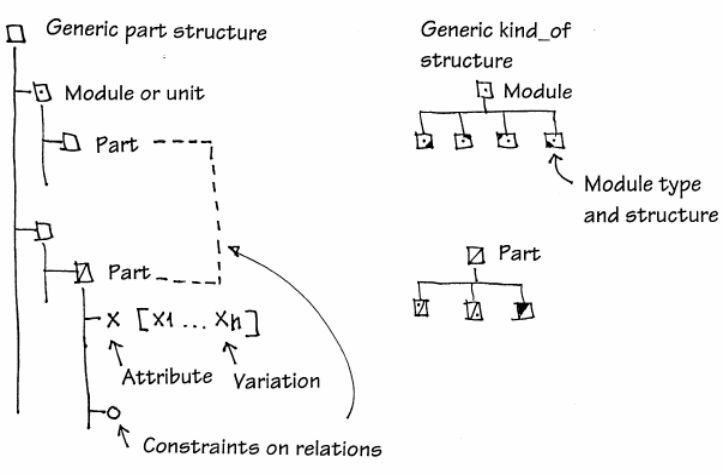


Figure 4: Contents of Product Family Master Plan [4,15]

A PFMP can be a strong tool when modelling products that exhibit a certain degree of kinship. For defining a product with few common characteristics the notation technique has some limitations, both concerning expression of relations between modules/units/parts and in that it does not deal with interfaces in an explicit manner. In some PCS research [17,18] the concepts **resources** and **ports** are used. Resources specify (functional) characteristics that components in the system can either supply or use and allow the specification of producer-consumer relationships among objects. Ports are places through which an object connects and communicates with other objects in its environment. When describing product families much of the structure is relatively stable, why interfaces of elements are only dealt with on a limited basis. Dealing with these situations using the PFMP technique can be done by representing the mentioned concepts in attributes that are linked together by constraints. In cases where one wishes to model products with a low degree of kinship, the PFMP is probably not the technique to use.

2.2 The PFMP in other product views

According to Hvam et al. [4] the PFMP can be applied for modelling of product meetings with life phase systems. Several examples are given in this book, but these do not include value domains of attributes or kind-of variation, which seems to be relevant in many cases. Riis [13] also states that the PFMP technique can be applied when modelling the product's meetings with life phase systems, though he acknowledges that the experience from this is very limited and recommends that one probably should use another notation technique for this purpose. On the other hand, Riis [13] claims that the PFMP is a strong modelling technique for the modelling of functions, organs and parts.

In figure 5 is an example of the division of a PFMP, where the model domains can be placed on the same or on different sheets during modelling.

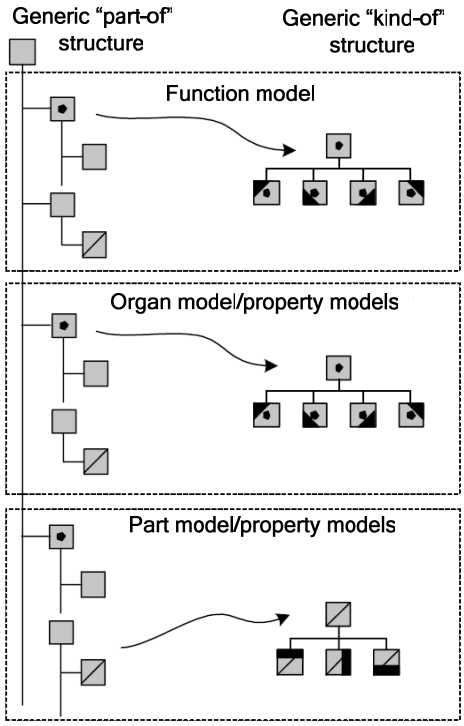


Figure 5: Example of the division of the PFMP [13] (Translated from Danish)

3 EMPIRICAL INVESTIGATIONS

3.1 Research method

The purpose of the empirical investigations was to discover possible problems or shortcomings when applying the definition of the PFMP in praxis. The investigations partly builds on the research project 'Product Models, Economy, Technology, and Organisation' (PETO), in which the co-author participated. The PETO project had the purpose of studying the process and effects of implementing PCS and included twelve Danish firms. For a more extensive description of the project see e.g. [19]. Based on these data, further interviews were conducted with seven persons from four of these projects. The questions that were asked concerned experience gained from applying the descriptions of how to use the PFMP in praxis. Since the purpose was not to compare the experience of different companies, but rather to explore the use of the PFMP technique, the form of the interviews was un-structured and semi-structured. Finally, observations of five modelling sessions in two ongoing PCS projects (not included in the PETO project) were carried out. The modelling sessions involved model-managers and domain experts who discussed and refined their product data model, modelled by using the PFMP technique and presented on printouts. The observer (the author) had the status of being an expert in the subject and the observations included some degree of participation in that the observer answered questions and suggested the use of alternative notation elements. After the modelling sessions, participants from these two projects were interviewed.

3.2 Results of research

The research carried out showed that there were several aspects of the existing definitions of the PFMP where a further development could be beneficial:

1. Need for extended formalism of notation
2. Solution of possible inexpediencies when using kind-of structure
3. Better possibilities of graphically displaying constraints on relations
4. Preparation of the notation for IT-supported modelling

5. Increased utility and instruction in applying notation for modelling the product through life phase systems

1) The performed interviews and observations indicated that some users of the PFMP could benefit from further formalism being added to the notation technique. For instance, two observations showed great difficulties for domain experts in understanding the generic aspect of the PFMP. The suggestion from the observer of displaying cardinality to some extent seemed to solve this issue. Another observation was that during some modelling sessions the model-managers were handed information, about which attributes that should be editable for the users of the planned PCS. This was not registered though this information would be needed at a later stage of the project. Subsequent interviews suggested that this was due to the fact that this element is not part of the formalism of the PFMP.

2) Some of the interviewed model managers stated that they found the way of displaying attributes of kind-of elements a bit elaborative due to the redundancies in the display of attribute names or lack of connection between values and kinds. This is illustrated in figure 6, which represents observed principles of use of the PFMP.

In principle (a) in figure 6 it is unclear which values are related to which type. This could obviously be stated in tables elsewhere, but doing this would imply that the information is scattered. In principle (b) it is clear which values belongs to which type. The downside of this principle is that the name of the attribute has to be stated repeatedly for each type.

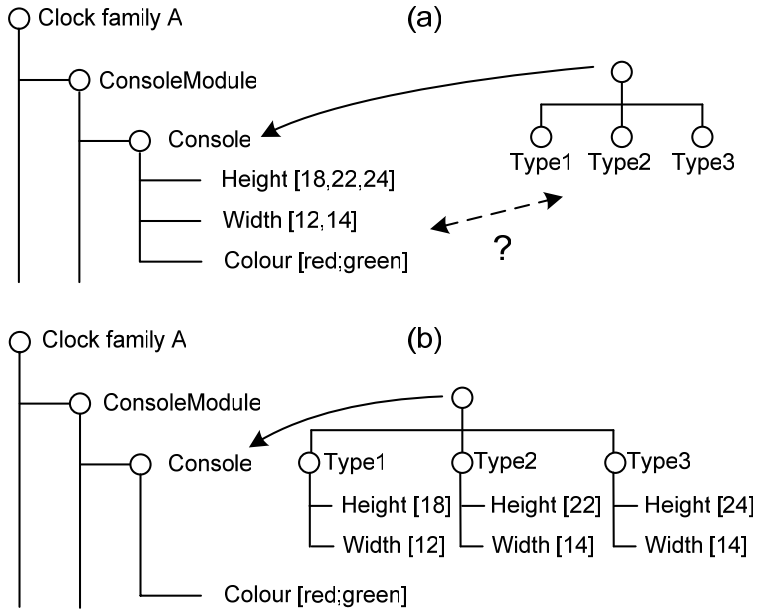


Figure 6: Principles for displaying kind-of structure

3) By graphically displaying relations between parts/attributes (as done in figure 4, termed **constraints on relations**) it becomes very visible which parts/attributes are connected by constraints. This aspect is relevant both during the creation of the PFMP and in later updates of the PFMP as documentation, where changes in one place can affect other parts of the model. The downside of graphically displaying a great number of relations is that this can take up a lot of horizontal space and make the printout of a PFMP confusing, as stated by some of the interviewed.

4) Several of the interviewed companies expressed interest in the planned IT-based modelling and documentation system. In order to include the PFMP in this system, extended formalism of the notation would be required to ensure that it is possible to model case-dependent types of information and that the PFMP models can be linked to other included notations. Documentation systems for PCS do exist though, but with much simplified use of the notation. This is e.g. described by Hvam and Malis [20] concerning a documentation system developed in Lotus Notes. In this system the user interface consists of a tree-structure, displaying components and corresponding CRC-cards. Compared

to the definition of the PFMP, the documentation system neither includes display of attributes or kind-of elements on the tree structure. So even if this principle seems adequate for documentation purposes, more information on the tree structure is needed in a modelling context, unless it is accepted that the product description is scattered over a high number of different sheets (CRC-cards).

5) The requirements of increased utility and instruction in how to apply the PFMP for modelling products through life phase systems was prior to the company studies forwarded by Riis [13]. The studies confirmed this view. Further, there seemed to be some redundancy in the representation of information when applying the PFMP for modelling products through life phase systems.

The fulfilment of the five mentioned requirements to a new notation technique should preferably not be at the expense of the learnability of the technique, since this was the main argument for using the PFMP technique instead of class diagrams. The argument of the learnability of the notation was confirmed by the investigations. Despite the expressional limitations and relatively inflexible structure compared to class diagrams, the aspect of learnability was weighted higher by many of the interviewed model-managers. For the evolved notation formalism it is therefore of paramount importance that the learnability aspect is maintained.

4 TOWARDS A NEW NOTATION TECHNIQUE

Before addressing the arguments for defining a new notation technique based on the PFMP, the existing theoretical frame, in where possible solutions should reside, is discussed.

4.1 Discussion of the existing theoretical frame

An important aspect to bear in mind, when looking at the existing theoretical foundation of the PBPCS, is that these theories are mainly targeted at product design (synthesis), whereas when constructing a PCS the main focus is on analysis of existing designs. There is an important difference between analytic models and synthetic models. An analytic model serves as a description, where a deliberate simplification is made (abstraction). In contrast, synthetic models are not a description of something existing, but instead created as a foundation for construction of something physical, an artefact [21]. This aspect could imply that other views of domains would be more appropriate in a PCS context.

Another important aspect is the distinction between different classes of models. Duffy and Andreasen [22] distinguish between three classes of models, moving from reality through phenomenon model, trough information model to computer model. Phenomenon models are based on design theories, as e.g. theory of technical systems. Information models are based on information theory, such as object-oriented modelling. Computer models are based on computational theories or languages. When describing a product in the context of building a PCS, we are moving from phenomenon models towards information models, which further implies that other model definitions than the ones from design theories could be more appropriate.

In the example of a product model divided in different views in figure 5 it is seen that: (1) internal and external property models are described together with the part domain and the organ domain respectively; (2) the function domain and organ domain are placed on different levels in contrast to the revised chromosome model in figure 2 and (3) models of meetings between the product and its life phase systems are not included

Looking at the material produced in the investigated PCS projects it was in some cases seen that some external properties as e.g. law conformance and design properties were presented in the PFMP as separate logical structures, in contrast to being placed as properties of parts or organs. This could suggest a need for modelling specific properties in a separate domain.

From a designer’s point of view the function of the product is the superior property, and if this aspect is not fulfilled the design is purposeless. Take for instance a simple product as a chair, even though the consumers might have their focus on properties such as aesthetics or ergonomics, these aspects lose their meaning if the chair does not fulfil its main function **allow persons to sit in.** Therefore, the designer first of all has to be aware of this aspect. From a configuration point of view this might not always be the case. If the aim is to describe how existing chairs can be combined by using different components, it seems reasonable to assume that these existing solutions are valid, i.e. allow persons to

sit in. This suggests that other superior properties than functions could just as well determine the physical structure of a product in a configuration context.

The view presented in the revised chromosome model where functions and organs are placed on the same level also seems to be sensible from a product configuration point of view. The structure of the function and the organ models would be partly similar if modelled in the same level of details. Obviously, different kinds of organs could realise the same function, but still there would be organs attached to each function and visa versa.

It seems to be desirable to have an efficient notation for modelling the meeting between a product and its life phase systems (life phase models) in the definitions of the notation technique. Modelling this aspect in other notations could complicate the creation of connectivity between the descriptions of a product and descriptions of its life phase models. As mentioned the argument for using the PFMP instead of other notations is primarily learnability. This aspect also supports the argument of being able to apply the notation in this context.

In this paper we do not wish to present a finite definition of the model domains needed in PCS projects, instead figure 7 shows a possible division. It should be noticed that some properties (including functions) reside in components or life phase model elements, and in many cases only superior properties (including functions) need to be modelled in the top domain. Another thing to observe is that the life phase models of the product are placed on the bottom level. This is done based on the argument that in a PCS context the product composition normally would decide the dimensions of these meetings, just as superior properties often would determine or restrict the physical composition. When creating the models of a project, only the relevant domains should be included, just as there can be several models (possibly on different levels) within a model domain. Furthermore, the models of the different domains do not have to be modelled on the same sheet, as it is shown in figure 7.

As seen in figure 7, we have replaced the term **part** by the term **component**. When defining the physical structure of a product, this does not necessarily mean dealing with machine parts. An element, which from a configuration perspective is perceived as indivisible, can consist of several machine parts in reality. Furthermore, the term **component** seems to have a more unambiguous meaning and a wider use within product configuration research. We define a component as being the smallest element of a product to be combined with others, from the point of view of the current company. For instance, if a car producer uses a prefabricated **engine**, this is considered a component of the product **car**. From the point of view of the company that manufactures the engine, the **engine** is the product, which is composed of a selection of components.

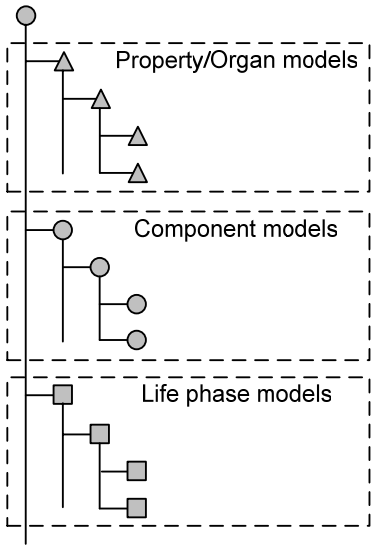


Figure: 7: Possible division of product models of a product configuration system

4.2 Modelling generic component structures

In this section we address the first four requirements for a new notation technique as described in section 3.2. This is done by proposing a new notation technique, which is shown in figure 8. Based on the extensive differences of the new notation formalism compared to the PFMP (which we still believe to be an efficient notation in many contexts) and to avoid mix-ups we have named the new notation technique **Product Family Diagram** (PFD). In the definition of the PFD it is chosen not to use the terms **module** or **unit** (as on figure 3) for describing groupings of components. This is mainly based on the argument that it in our context a component itself can represent a unit or a module. Instead, the term **sub-structure** is chosen, which can represent an actual assembly or a logical grouping of components for structural reasons. The shown notation formalism in figure 8 can also be applied on property and organ models in accordance with figure 7.

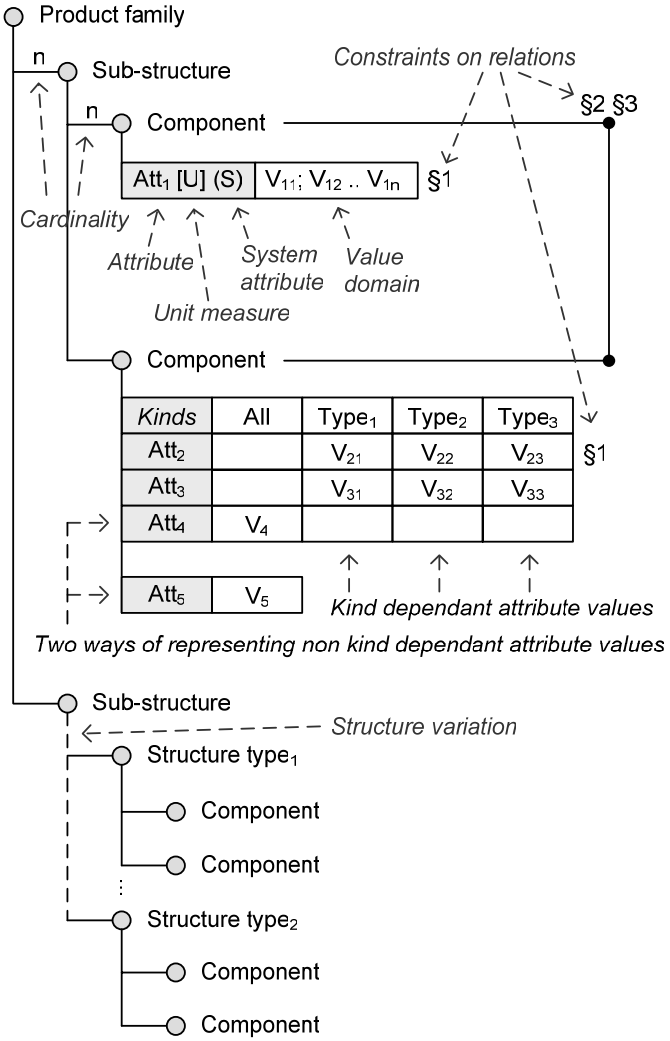


Figure 8: Notation principle for a Product Family Diagram for components

1) To comply with the requests for increased formalism of the notation the following elements have been introduced:

- Display of cardinality
- Unit measure
- Attribute type
- Constraint reference

The notation formalism of the PFD includes display of cardinality in order to visualise that a component/sub-structure can have more or fewer instances than one. The cardinality number **n** can be expressed as both numbers and intervals, e.g. 1 to 5, expressed as '1..5'.

The formalism for representing attributes has been enhanced by adding **unit measure** and **attribute type** to the notation. The suggestion of how to display a system attribute (not modifiable by the users) is only instructive. If the user of the notation technique wishes to illustrate other types of attributes (e.g. hidden), one can do so by using an individual choice of symbolism.

As shown in figure 8, the notation formalism of the PFD includes two ways of displaying constraints on relations, by using lines with references (as for §2 and §3) or by just stating a reference behind elements of the PFD (as for §1). Constraints on relations can be placed both on sub-structure, component and attribute level.

In the definition of the PFD (figure 8) the term **constraints on relations** is used. The term should in this context be understood in a broad sense, opposed to referring specifically to constraint-based reasoning. The PFD can just as well be applied when using other kinds of reasoning, as e.g. rule-based. In literature there exist different ways of classifying reasoning, for further descriptions of this topic see e.g. [17,18,23,24].

2) To solve the problem relating to the way attributes are shown on kind-of structure (seen on figure 6) two possible notation principles for type dependant variation were considered, as shown in figure 9.

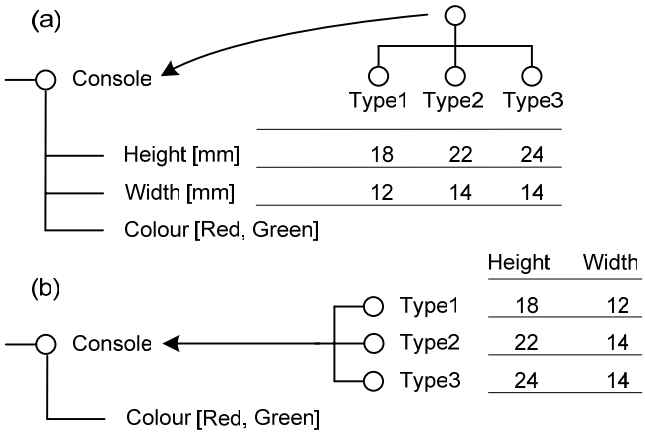


Figure 9: Possible principles for assigning values to types

On both the illustrated principles in figure 9 it now clearly appears which attribute values belong to which type, without having to type names of attributes more than once. Principle (a) was chosen primarily based on the argument that attributes are then shown in the same direction in both part-of and kind-of structure.

Besides solving the earlier mentioned aspect of linking attribute values to kinds, the existing notation for displaying kind-of structure and attributes was replaced with tables as seen in figure 8. This aims at contributing to a clearer overview of the diagram and is further based on the assumption that most domain experts are familiar with tables.

3) As seen in figure 8, the part-of structure has merged with the kind-of structure. This aspect to some extent solves the issue of displaying constraints on relations, as the lines between components/sub-structures are not crossing the kind of structure. This is illustrated in figure 10 and figure 11, which show the same example of a generic component model using the existing (PFMP) and the new notation technique (PFD) respectively.

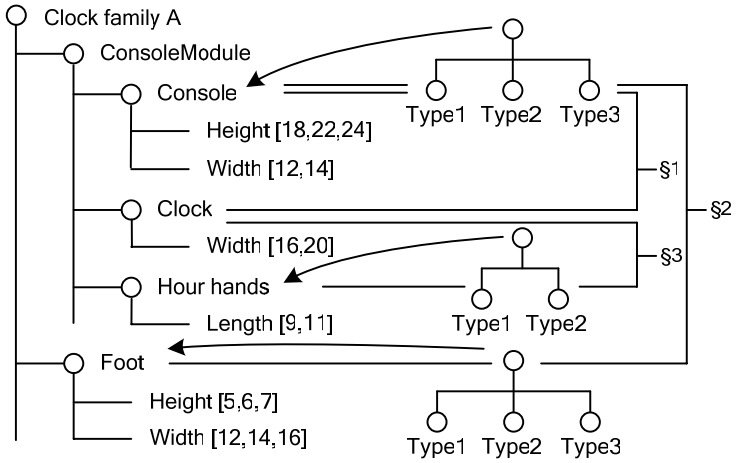


Figure 10: Product with relations using the PFMP notation

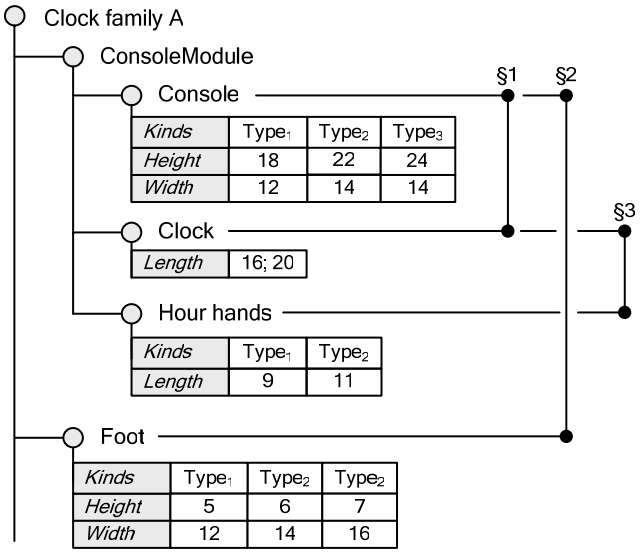


Figure 11: Product with relations using PFD notation

4) By merging kind-of structure with part-of structure for components, more control of the placement of the kind-of elements is achieved, which would make the construction of an IT-based modelling tool easier. The same goes for variation in sub-structures, which have also been moved to the part-of side of the diagram. Modelling an unpredictable number of sub-structures on the kind-of side could present some problems in an IT-based modelling environment as regards automatic placement of the different elements of a PFD. We recognise that some learnability could be lost through this modification, though we at this point have no indication of such.

An issue that was raised when the suggested notation technique was presented to model-managers was how to model kind-of structure for components with much different attributes. To deal with this the two principles shown in figure 12 can be applied. On (a) the values for uncommon attributes are left blank, while the kind-of variation in principle (b) is treated as structure variation.

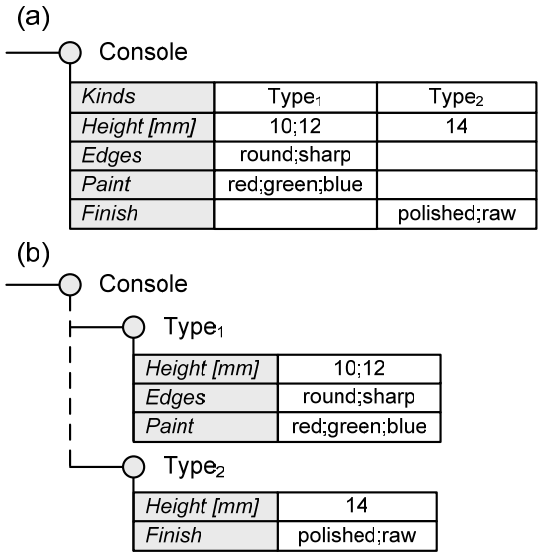


Figure 12: Principles for modelling kinds with different attributes

4.3 Overall principle of a product family diagram

Having defined the notation formalism for describing product structure we now move one level higher and define the overall structure of a PFD. We define a PFD as consisting of four layers, which would normally all be relevant, at least when analysing the component structure. The four layers are shown in figure 13, where constraints can either be drawn on the PFD, listed in separate sheets or described in CRC cards.

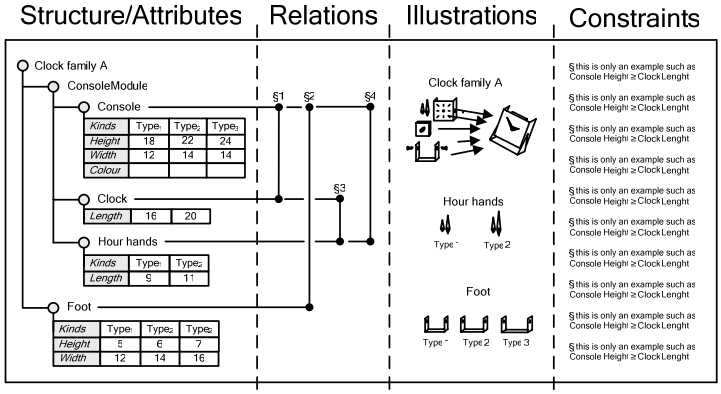


Figure 13: The four layers of a Product Family Diagram

The use of the term **layers** has another perspective than just division of content, as it in the context of IT-based modelling support the possibility of turning layers on and off for user defined views.

In the maintenance phase some model-managers might prefer to maintain each component/sub-structure individually on CRC cards, where attributes, relations, illustrations, constraints and more information can be found. In this setting only the component hierarchy of the PFD needs to be shown for overview and navigation. This is shown in fig 14.

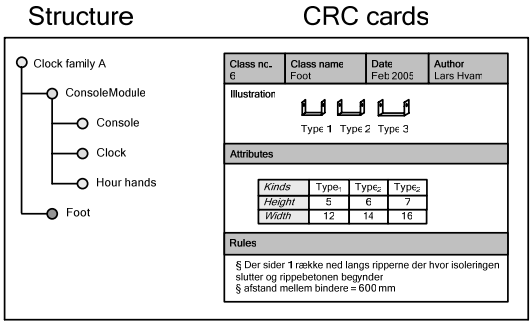


Figure 14: Maintenance of product model

4.4 Product life phase models

As mentioned, the theory of dispositions [12] links the decisions concerning the design of a product to the effects that these dispositions have during the life-phases of the developed artefact. In the same way the configuration activity, where different components are chosen, has an impact on the life phases of the product. This aspect gives basis for the concept of generic life phase models that based on a specific component configuration automatically can produce the resulting instances of the relevant life phase models.

The types of information, embedded in different life phase models, can vary depending on type and focus of the PCS. In this paper we choose to focus on life phase models that could be part of a PCS for sales, and to a lesser degree manufacturing. The relevant types of information about the life phases of a product in a sales context could concern aspects such as price, time and required resources. This kind of information comes from processes like fabrication of non-prefabricated components, assembly of components and delivery of the product. These processes can be decomposed into sub-processes that share common characteristics, but different dimensions. In an object-oriented terminology this can be formulated as: the sub-processes, which constitute the process of a life phase model, are objects from the same class, i.e. specialisations from a general class. Since sub-processes more or less would have the same attributes, application of the existing notation when creating a model implies that the same names of attributes would be presented repeatedly in the representation. This problem is illustrated in an example on part (a) of figure 15, which represents an often seen principle from students who applies the PFMP technique for describing operations. As seen, operation 1 is dependent on the choice of component-type, illustrated with kind-of structure.

Of the previously discussed possibilities of showing kind-of structure (figure 5) only the vertical listing of kinds seems to solve the problem of repeated attribute names. This is included in the proposed notation technique **PFD for life phase models**, shown on part (b) of figure 15. When comparing part (a) and (b), it is seen that the proposed notation formalism delivers a much more compact representation by eliminating the redundant display of attribute names and units of measurement.

When using the PFD for life phase models an operation could have an individual attribute. In this case another column can be added, which is left blank for operations not including this attribute. In other cases a constraint reference is needed instead of a value domain. The expressions of these constraints can also be placed in a column.

Assuming a product relating to the life phase model in figure 16 has been configured. If $type_1$ of $component_1$ is chosen, this implies that the total time of the operations equals 46 minutes, and the cost is 40 Euro, if $type_2$ was chosen respectively 48 minutes and 45 euro, and so on. This illustrates in a simple way that the dimensions of a life phase model of a product can be derived from the dispositions of the product configuration activity.

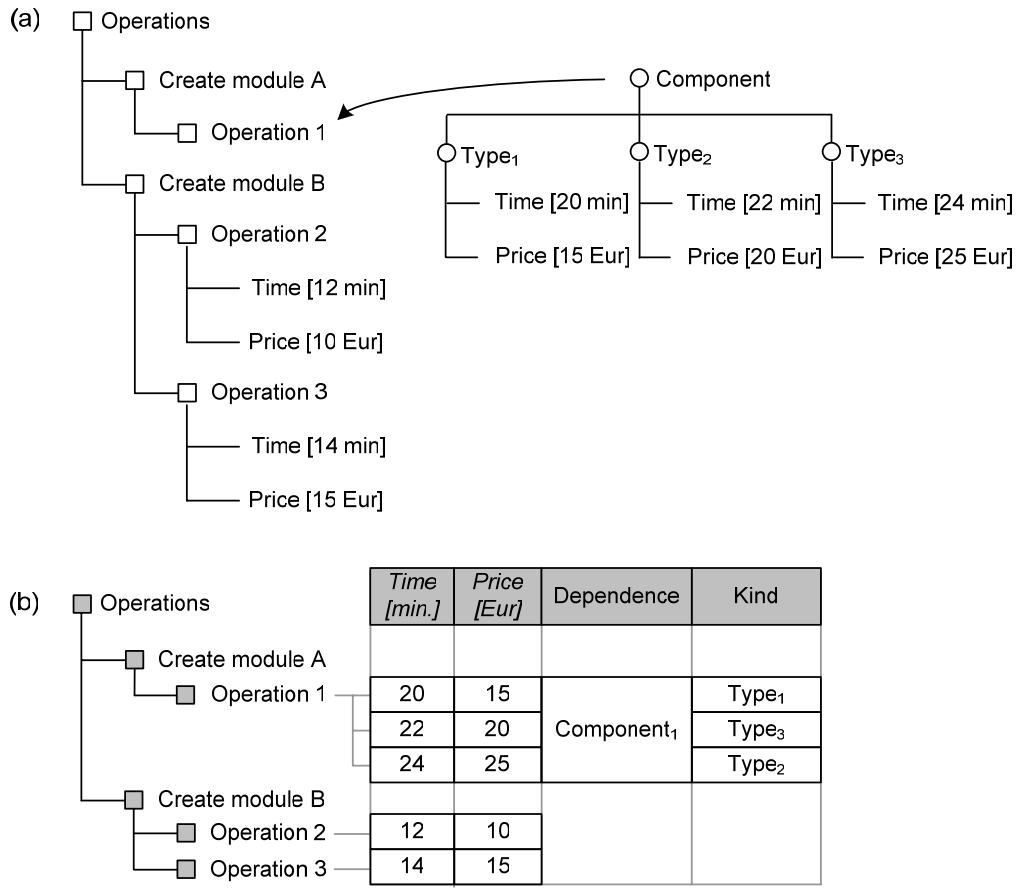


Figure 15: Example of the PFMP applied on operations

5 CONCLUSIONS AND FURTHER RESEARCH

In this paper the existing way of perceiving product models in connection with the PBPCS was analysed. This led to a proposition of a revised division of model views to be used when analysing and describing products families, as a step towards the construction of a PCS. The main suggestions concerned an extension of the function view to include other properties and organs.

A new notation technique for elicitation of product knowledge was presented, termed **Product Family Diagram**. The notation is based on the existing PFMP [15] but with great changes of formalism and embracing diagram types for specification of both component structure and life phase models.

The new PFD diagrams addresses five groups of issues related to the use of the existing PFMP technique, which were discovered through studies of company projects:

1. Need for extended formalism of notation
2. Solution of possible inexpediencies when using kind-of structure
3. Better possibilities of graphically displaying constraints on relations
4. Preparation of the notation for IT-supported modelling
5. Increased utility and instruction in applying notation for modelling the product through life phase systems

The proposed PFD technique has yet to undergo tests in company projects, and is not at its current state considered to be final. The testing of the PFD technique is firstly done by using the technique instead of the PFMP when applying the PBPCS in company projects. This is planned to be carried out before the end of 2005. Furthermore, the PFD technique is planned to be part of the first prototypes of an IT-based modelling and documentation system, which some companies have already agreed to test.

It is our belief that these experiments will provide the necessary basis for evolving the proposed notation technique to an even more applicable and well-defined state.

6 REFERENCES

[1] Hvam, L., 1994, Application of product modelling – seen from a work preparation viewpoint, PhD thesis, Department of Industrial Management and Engineering, Technical University of Denmark.

[2] Hvam, L., Riis, J., Malis, M., 2002, A multi-perspective approach for the design of configuration systems, 15th European Conference on Artificial Intelligence, Lyon, France.

[3] Hvam, L., 2004, A multi-perspective approach for the design of Product Configuration Systems - an evaluation of industry applications, International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems, Technical University of Denmark.

[4] Hvam, L., Mortensen, N.H., Riis, J., 2004, Produktkonfigurering, (In Danish), Publication used for teaching product configuration, Department of Industrial Management and Engineering, Technical University of Denmark.

[5] Hansen, C.T., Andreasen, M.M., 2002, Two approaches to synthesis based on the domain theory, in Engineering Design Synthesis, (ed. A. Chakrabarti), Springer-Verlag, chapter 6: 93-108.

[6] Hubka, V., Eder, W.E., 1984, Theory of Technical Systems, Springer-Verlag, New York.

[7] Andreasen, M.M., 1980, Machine Design Methods based on a Systematic Approach – Contribution to a Design Theory, (In Danish), Dissertation, Department of Machine Design, Lund University, Sweden.

[8] Hubka, V., Eder, W.E., 1988, Theory of Technical Systems, Springer-Verlag, Berlin.

[9] Andreasen, M.M., 1992, The Theory of Domains, Working paper, Institute for Engineering Design, Technical University of Denmark.

[10] Mortensen, N.H., 1999, Design modelling in a designers workbench, PhD thesis, Department of Control and Engineering Design, Technical University of Denmark.

[11] Andreasen, M.M., McAloone, T., Mortensen, N.H., 2001, Multi-Product Development-platforms and modularization, Technical University of Denmark, Lyngby.

[12] Olesen, J., 1992, Concurrent Development in Manufacturing - based on dispositional mechanisms, PhD thesis, Institut for Engineering Design, Technical University of Denmark.

[13] Riis, J., 2003, Fremgangsmåde for opbygning, implementering og vedligeholdelse af produkt-modeller - med fokus på konfigureringssystemer, (In Danish), PhD thesis, Department of Manufacturing Engineering and Management, Technical University of Denmark .

[14] Miller, T.D., 2001, Modular Engineering, PhD thesis, Department of Mechanical Engineering, Technical University of Denmark.

[15] Mortensen, N. H., Yu, B., Skovgaard, H. and Harlou, U., 2000, Conceptual modeling of product families in configuration projects, 14th European Conference on Artificial Intelligence, Configuration Workshop, Berlin, Germany.

[16] Hvam, L., Riis, J., 1999, CRC-Cards for Product Modelling, 4th Annual International Conference on Industrial Engineering Theory, Applications and Practice, San Antonio, Texas.

[17] Soininen, T., Tiihonen, J., Männistö, T., R. Sulonen, 1998, Towards a General Ontology of Configuration, AIEDAM, Special Issue on Configuration, 12/4: 383-397.

[18] Sabin D., Weigel R., 1998: Product Configuration Frameworks – A Survey, IEEE Intelligent Systems & their applications, 13/4:42-49.

[19] Edwards, K., Møldrup, M., 2004, Unexpected emergence of a Community of Practice when implementing Product Configuration Systems**,** 9th International Conference - Human & Organisational Issues in the Digital Enterprise, National University of Ireland, Galway.

[20] Hvam, L., Malis, M., 2001, A Knowledge-based Documentation Tool for Configuration Projects, World Congress on Mass Customization and Personalization, Hong Kong.

[21] Jørgensen, K.A., 2004, Modelling on Multiple Abstraction Levels, 7th Workshop on Product Structuring, Chalmers University of Technology.

[22] Duffy, A.H.B., Andreasen, M.M., 1995, Enhancing the evolution of design science, 10th international Conference on Engineering Design, ICED95, Prague.

[23] Stumptner, M., 1997, An overview of knowledge-based configuration, AI Com, 10/2: 111-126.

[24] Jackson, P., 1999, Introduction to expert systems, third edition, Addison-Wesley Pub Co.

Paper B2

A comparative study of two graphical notations for the development of product configuration systems

Haug, A. and Hvam, L. (2007): "A comparative study of two graphical notations for the development of product configuration systems", International Journal of Industrial Engineering, Vol. 14, No. 2.

A comparative study of two graphical notations for the development of product configuration systems

Anders Haug and Lars Hvam

In this article two graphical notations for modelling product knowledge as a basis for building product configuration systems are compared. The two notations in focus are respectively class diagrams of UML and the so-called product family master plan. When choosing a graphical notation for describing the product knowledge to be implemented in a configuration system, at least two aspects should be considered, the expressional strength and the usability of the notation. The analysis of expressional strength is done by comparing the two notations in relation to possible modelling scenarios. The usability comparison is based on six studies of product configuration projects. The comparative study of the two notations clarifies their strengths and weaknesses and can be seen as a help to project managers of product configuration projects by supporting their evaluation of which graphical notation to apply in a specific context.

Significance: To make a well-founded choice of graphical notations to be used in a product configuration project, a clarification of their strengths and limitations is needed. This article provides this basis by investigating two commonly used graphical notations for modelling product configuration knowledge.

Keywords: Product configuration, product modelling, knowledge representation, product family master plan, class diagram

1. INTRODUCTION

The use of product configuration systems (PCS) is a technology that supports the manufacturing paradigm of mass customisation (Pine et al., 1993). Several definitions of PCS's have been produced. In this article the term PCS is used for describing software systems that can generate specifications of product variants that fit particular needs by combining sets of pre-defined components according to restrictions on how the components can be combined. The use of such a PCS to specify products of a certain complexity and commonality allows companies to achieve a high degree of product variance while keeping the costs of specifying the products low.

A PCS includes a knowledge base, which can be perceived as a generic product model that determines the legal instances of the product. This allows PCS's to perform tasks that earlier were carried out by human experts. The relocation of knowledge from domain experts to an IT-system creates a need for elicitation of product knowledge. According to Sabin and Weigel (1998), most of the complexity in solving a configuration problem concerns the representation of domain knowledge. To support elicitation of domain expert knowledge and design of the knowledge base of a PCS, several approaches, which include different graphical notations, have been proposed. While approaches provided by international researchers often rely on UML class diagrams, some Danish approaches apply another graphical notation, the so-called product family master plan (PFMP). PFMP's are far less internationally known and has several limitations compared to class diagrams. The main arguments for the use of this notation focus on the learnability and efficiency, which the notation is claimed to possess when creating conceptual product models.

As most approaches for developing PCS's do not clearly specify demarcations or limitations in respect to the contexts of their potential use, companies standing before the introduction of a PCS-project are in a poor position to choose between different modelling techniques. By elucidating the strengths and weaknesses of PFMP's and class diagrams, this article aims at providing a more solid basis for choosing the notation that best supports a given modelling task.

This article is primarily written with the use of standard configuration system software in mind, which means that only the part of PCS development that concerns the knowledge base is included, contrary to design of other PCS features. Nevertheless the article would still be relevant when dealing with custom-made PCS's.

The article is structured as follows. In section 2 class diagrams and PFMP's are described. Next, in section 3, the expressional strengths of the two notations are compared. In section 4 usability aspects of the two notations are investigated, based on empirical studies of their use in product configuration projects. The article ends with a conclusion in section 5.

2. NOTATIONS FOR DEFINING THE KNOWLEDGE BASE OF A PCS

Within software development there is a need to manage the complexity in both analysis of the real world and in design of software systems, for which models provide abstracted views by primarily containing high-level information. Some models can help to understand the area addressed by the system before the activities of system design and coding are initiated. These models are referred to as *analysis models* (also called *conceptual models* and *domain models*) and do not directly refer to the properties of the software system, contrary to *design models* (Priestley, 2003). In the development of a PCS, analysis models can be created to define the product knowledge that is to be included in the PCS. Much of this knowledge typically resides in the heads of domain experts why the challenge is to elicit this knowledge. When the analysis model is created, it must be considered how this knowledge should be represented in a PCS, which is captured in a design model. A design model, for instance drawn in class diagrams, describes the architecture of a program at a higher level of abstraction than source code. Such models can be an invaluable help for software engineers (both developers and maintainers) to analyse programs architectures, design choices, behaviours, and implementations (Guéhéneuc and Albin-Amiot, 2004). Differentiating between the two kinds of models does not mean that there necessarily have to be big differences between these. In PCS development, design models would often be refined or extended versions of analysis models.

For some assignments graphical notations are good, for others text works better. Graphical notations are well suited for displaying structural aspects of a system, but less effective for documenting detailed properties of model elements or the restrictions that might be placed upon them by a *business rule* (Priestley, 2003). Another thing to consider is how close a notation is mapped to the problem domain, where a closer mapping would require use of fewer lexemes. The choice of notations for creating analysis and design models obviously affects how information is represented, but also the course of the modelling processes. A typical trade-off concerns the fact that some notations are more user-friendly than others but in return have expressional limitations compared to more complex notations.

When starting a PCS project, where a product family indicates modularity as a basis for applying configuration systems, it often has no well-considered or permanent architecture as a basis for creating a robust generic product model (Pulkkinen, 2000). Therefore, a product family should be structured for configuration before the actual development of a PCS is begun. This urges agreements between the product experts and the PCS modelling team on issues like scope of the PCS, compositional constraints of the products, customisable properties of the products etc. This process requires a common language for discussing and capturing decisions on the generic aspect of the product. In this context models made in graphical notations often provide good overviews.

Many domain experts can be involved in the elicitation of product knowledge as a basis for building a PCS, where some are only exposed to the knowledge representations in shorter periods of time. This urges the use of a graphical notation of adequate low complexity in order to avoid extensive training of domain experts. On the other hand, a very simple graphical notation might have expressional limitations, which can make it problematic to capture all relevant information. Therefore, when choosing a notation, at least two aspects have to be considered, namely how much learning would the users of the notation require, and how well does the notation support the specific modelling assignment. In this context also the configuration software plays an important role, since it determines the final limitations of what is possible to implement.

2.1. UML Class Diagrams

UML is an object-oriented modelling language, not a method nor a methodology. In 1997, UML was officially adopted as a standard by the Object Management Group (OMG). The enticement to create UML was that there in the early 1990s were several competing visual modelling languages and methodologies, where UML was meant to unify the best elements from these methods. The UML specification in its current state consists of two interrelated parts, the UML metamodel and the UML notation. The UML metamodel specifies the UML concepts, while the UML notation is the graphical syntax of the modelling language. The UML metamodel is based on Meta Object Facility (MOF), which is also used as metamodel for other languages (OMG, 2005).

In UML 2.0 there are thirteen types of diagrams, which are divided into three categories: six s*tructure diagrams*, three *behaviour diagrams* and four *interaction diagrams* (subtype of behaviour diagram) (OMG, 2005). In this article one of the static diagram types is in focus, namely the *class diagram*. This does not mean that other UML diagrams would not be useful in a product configuration context, on the contrary. But in spite of this, the class diagram must be said to be far the most important UML diagram for the modelling of product configuration knowledge, as this knowledge is most often described in terms of elements and their relationships.

A class can be defined as a set of objects that share common features, where objects cannot exist without a class. Class diagrams depict the static structure of a system, which includes classes, their features and relationships between the classes. In UML terminology *features* refers to the *properties* and *operations* of a class. The *properties* are the structural features that appear in two distinct notations, *attributes* and *associations*, which more or less are used to express the same thing (Fowler, 2005). *Operations* are the actions that a class knows how to execute, which therefore determine the behaviour of a class.

The notation for the class element and the most common relationship types are shown on figure 1. Furthermore, a *navigability arrow* can be used to show the direction of association, aggregation and composition relationships.

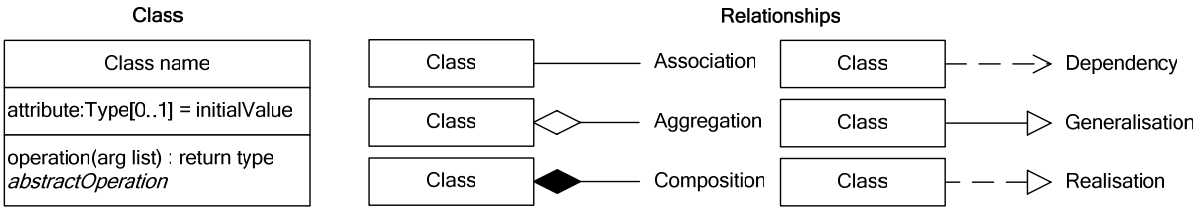


Figure 1: Elements of class diagrams

In a PCS context, expressions of constraints regarding how elements can be combined are essential. UML provides a formal constraint language called OCL (Object Constraint Language), but any formalism to describe constraints are allowed as long as placed within braces ({}). A constraint string may be placed inside or near the model element that it describes, for instance following an attribute or on a relation between two classes. The constraints strings may also be placed in the UML note symbol and attached to model elements by a dashed line.

2.2. Class diagrams in a product configuration context

UML class diagrams include a broad set of model elements to support different kinds of applications within software development. Class diagrams can therefore be confusing if the users are presented to the full notation, which in most applications would also be to give the users more means than they need. As explained by Odell et al. (2000), it is worthwhile to define a modelling language that is a refinement of a well-known modelling language, since learning this one will be simplified. This idea has been applied to class diagrams to target it at modelling the knowledge of PCS's, where two different approaches are resumed in this section.

Among approaches, which apply class diagrams for developing PCS's, the one by Felfernig et al. (2000; 2001) is among the most prominent. The choice of UML in this approach is based on arguments of the widely appliance in industrial software development, and because of their good experiences with the use of UML designs for validation by technical domain experts. The key idea of the approach is firstly to extend the static UML model by commonly used configuration concepts, and secondly to create a definition of how these concepts are mapped to a configuration language (Felfernig et al., 2000). The mean for extending UML is by defining additional modelling concepts inside UML by introduction of a *profile* (Felfernig et al., 2001). Profiles were added to the UML 1.4 specification, and can be used to extend a reference metamodel with the purpose of adapting the metamodel to a specific platform or domain, by use of stereotypes that apply to existing metaclasses (OMG, 2005). In (Felfernig et al., 2001) the class stereotypes for the configuration domain are defined as *component*, *resource*, *function* and *port*, the specialised associations as *incompatible* and *is_connected*, and the specialised dependencies as *requires*, *produces* and *consumes*. Constraints that cannot be expressed graphically are formulated using OCL.

Configuration knowledge models often include both a structural and a functional architecture. In (Felfernig et al., 2001) these two views on a product are interrelated through a mapping between functions and physical components, which can be expressed through the dependency relationship or additional constraints.

Another approach to development of PCS's, which includes the use of class diagrams, is a procedure from the Centre for Product Modelling (CPM) at the Technical University of Denmark. The CPM-procedure was introduced by Hvam (1994) and has since evolved through (Hvam, 1999) to the definitions found in Hvam (2004). The procedure covers most of a PCS project by providing guidelines through its seven phases: 1 process analysis, 2 product analysis, 3 object-oriented analysis, 4 object-oriented design, 5 programming, 6 implementation and 7 maintenance. The procedure (or part of the procedure) has been tested in projects carried out by several companies (Hvam et al., 2002; Hvam, 2004).

The CPM-procedure has separate phases with different knowledge representation techniques for respectively describing the products in the scope (phase 2) and for creating a more formal definition of the knowledge base along with specification of other system features of the PCS (phase 3-4). The product analysis phase uses the PFMP notation, which is described in the next section. The object-

oriented analysis and design phases are supported by UML, primarily class diagrams, which are proposed for refinement of the structure of PFMP's from the preceding phase. The procedure attempts to capitalise from the use of both notations in the same developments process by benefiting from the learnability of PFMP's for elicitation of product knowledge from domain experts, while using the more formalised and flexible class diagrams for design of the knowledge base of a PCS. The procedure defines usage of a subset of the elements of class diagrams limited to the three relationship types: generalisation, aggregation and association (Hvam et al., 2002; Hvam, 2004). The definitions do not include the concepts of interfaces and resources nor do they include stereotypes. Instead, the use of special CRC-cards to allow further specification of class diagrams is proposed. Compared to the original CRC-cards (Beck and Cunningham, 1989), the adapted CRC-cards (Hvam and Riis, 2003) are extended by additional fields, such as *superparts/subparts*, *superclass/subclass, object mission*, *sketch* and *object knows/does* (attributes/methods).

2.3. Product Family Master Plan

The PFMP notation (Mortensen, 1999) is far less widespread than class diagrams, a part from its usage together with standard configuration systems, mainly in Denmark. The notation is in some contexts referred to as a *product variant master*, but in this article the first mentioned term is used. The PFMP notation is an important element of two procedures for building PCS's. The first one is the earlier described CPM-procedure. The original version of the CPM-procedure (Hvam, 1994) did not include PFMP's and instead class diagrams were proposed for use in knowledge elicitation from domain experts. The projects in which the early version of the procedure was applied faced several problems in making domain experts understand the notation of class diagrams, which led to that PFMP's were incorporated into the procedure.

The second procedure is by Mortensen et al. (2000) (Mortensen, 2001). This procedure does in contrast to the CPM-procedure only include the PFMP notation for the creation of structural graphical models, compared to also applying class diagrams. The procedure also suggests the application of extended CRC-cards, similar to the ones of the CPM-procedure. A PFMP basically consists of two generic sections, describing part-of structure and kind-of structure. The part-of section shows the whole-part hierarchy of the classes of the product, while the kind-of structure displays variation within different classes, i.e. specialisations. In figure 2 an example of the use of a PFMP is shown.

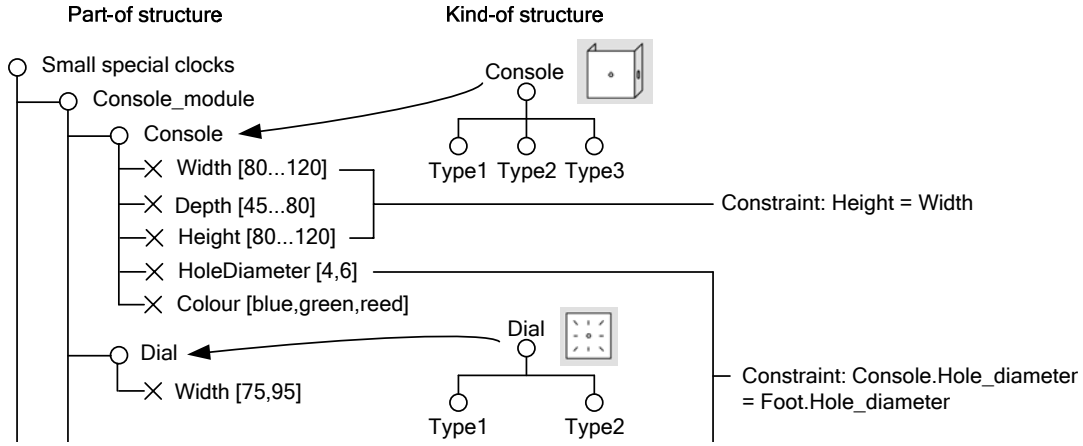


Figure 2: Example of PFMP

3. COMPARISON OF EXPRESSIONAL STRENGTH

3.1. Relationship types

A PFMP supports three kinds of relationships between its elements: *part-of, kind-of* and *constraint relations*, as seen on figure 2. According to Hvam (2004), the kind-of relationship type can be translated into *generalisation* and the *part-of* relationship into *aggregation.* In this context aggregation

is perceived in a broad sense, embracing both *aggregation* and *composition*. While the PFMP notation allows the expressing of generalisation, it has a limitation in that it does not allow graphical differentiation between the aggregation and composition. Both aggregation and composition are whole-part relationships, but they differ in that the composition relationship type requires that a part instance is included in at most one composite at a time, and that the composite object is responsible for the creation and destruction of the parts. When dealing with product compositions, the composition relationship seems to be the one to use, since parts of a composite normally only would belong to one composite. The differentiation between aggregation and composition is one of the most frequent sources of confusion in UML according to Fowler (2005), who goes as far as recommending to ignore the aggregation relationship type in software design.

In a PFMP the last relation type is the *constraint relation*, which cannot be said to correspond to a specific relationship type in class diagrams. The constraint relation of PFMP's can be drawn both between attributes of the same class, between classes in a whole-part relation, and between classes that do not have any prior relationship. In the case where there is a constraint between attributes within the same class of a PFMP, this would not be graphically displayed by a relation in a class diagram, but merely expressed as a constraint string belonging to the class. In the case where a constraint relation is drawn between elements of a PFMP, which are in a whole-part relationship, this would not in a class diagram correspond to creating a new relation, but instead to place a constraint string on an existing relation. This means that using a PFMP in this case requires two relationship types between the same classes in contrast to class diagrams where an existing relation is reused. In the case where two unrelated elements of a PFMP are connected with the constraint relation, this corresponds to the dependency relationship of class diagrams, i.e. a relationship where a change to one modelling element will affect the other modelling element. However, the PFMP notation does not prescribe the use of directions of constraint relations, wherefore it can be unclear which class should later hold the constraint.

The PFMP notation does not include relationship types corresponding to *association* and *realisation* of UML. An association relation could be expressed in a PFMP by using the constraint relation, but this would not allow differentiation between association and dependency relationships. In the given context, the use of the relationship type *realisation* (according to the experience of the author) does not seem to be a much needed concept to apply, as these generic product models do normally not deal with how parts of a product are realised in such an explicit fashion.

A more significant expressional limitation of PFMP's compared to class diagrams is the inability to model classes included in multiple wholes or classes that are both included in a whole and inherit from another class. The limitation concerning multiple wholes means that a component or sub-structure can appear at several places on a PFMP without it being expressed graphically that it is the same element. This is illustrated in figure 3, where also classes that are included in wholes and which inherit from a superclass are shown on the class diagram part.

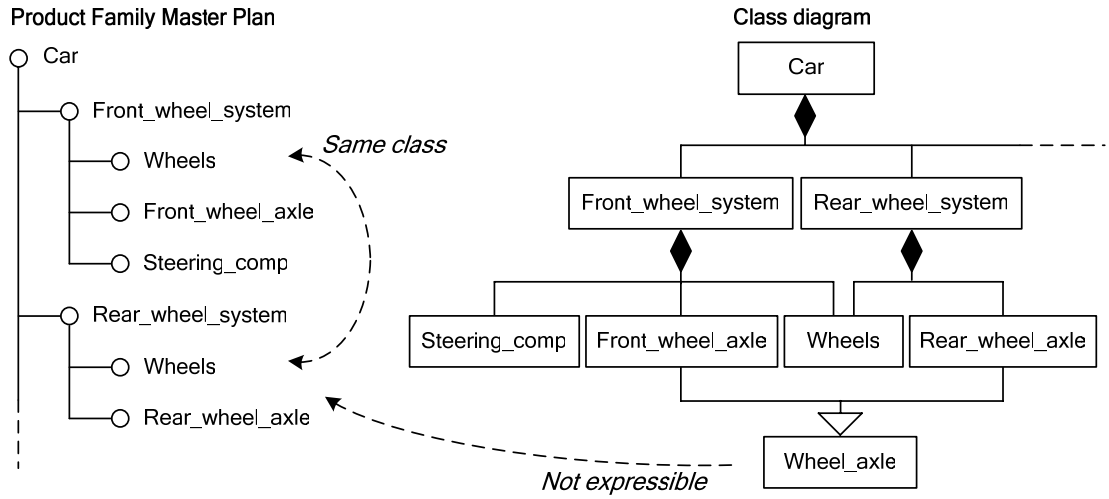


Figure 3: Expressing multiple wholes and inheritance

The inability to express multiple wholes in a PFMP could be solved by placing the repeated class of a PFMP outside the main structure and then make an external reference to this when needed. Another possibility could be use the constraint relation (or similar notation) to navigate the reader back to the first occurrence of the repeated element. The problem of expressing that a class both belongs to a whole and inherits from a superclass is even more problematic to express in PFMP's. If this must be expressed, a possibility is to show the same class at two separate places in the model, in a whole-part relation and as a specialisation in the kind-of section of a PFMP respectively. Despite of the possible bypasses, these aspects must be considered as significant limitations of the PFMP notation compared to class diagrams.

3.2. Other expressional aspects

As mentioned in section 2.2, Felfernig makes use of the stereotype concept on classes and relations to specialise class diagrams for describing product models. The use of stereotypes allows a visible distinction between the applied types of classes and relationships. In the PFMP notation this distinction is neither supported by predefined symbols nor by extension mechanisms. An obvious solution would be to apply the stereotype concept of UML to the PFMP notation.

As mentioned, the current version of UML (2.0) consists of thirteen diagram types. Having diagrams, which can be used in extension of class diagrams for modelling other structural aspects or behavioural and interaction aspects, would in many cases be advantageous. When using a PFMP no such related diagrams exist.

No standards exist for how to express constraints/rules in a PFMP. The language OCL is a part of UML and would in some cases be obvious to apply together with class diagrams, as using a formal language for expressing rules/constraints minimises the risk of misinterpretation. This, however, presumes that the users fully understand the language, if not an informal language might work better. OCL is based on predicate calculus, which makes it hard to understand for people without prior knowledge of this. In the scenarios where domain experts are involved in the modelling, OCL could turn out be to complex to use unless extensive training is provided.

4. USABILITY STUDIES

Based on the expressional limitations of PFMP's compared to class diagrams it could seem pointless using the first mentioned notation, but the PFMP has its strengths in its usability, which is being dealt with in this section.

Human-computer interaction (HCI) can roughly be described as a field that deals with computer systems that people interact with. Though not quite the same, there are significant similarities in requirements for a software system and a graphical notation to be used for modelling product knowledge. Also the common praxis of software supported elaboration of conceptual models should be considered in this context. Within HCI research several definitions of usability exist. In this article a basic distinction between *utility* and *usability* is used (Nielsen, 1993; Grudin, 1992). Utility concerns how well the design suits the (functional) needs of the user. The investigations of the expressional strengths of the notations in section 3 of this article can therefore be perceived as an utility analysis, given that the expressional possibilities of a notation is seen in relation to the requirements from the context where it is used. Usability concerns how easy an interface is to use. Nielsen (1992) describes five essential usability characteristics, which according to Holzinger (2005) are generally accepted as being part of any software project: learnability (how rapid users can be working with the system), efficiency (enabling users, who have learned to use the system, to attain high productivity), memorability (allowing casual users to return to the system after periods of non-use), errors (errors made by users and ability to recover) and satisfaction (how pleasant the system is to use). To investigate usability aspects of the two notations, the focus is firstly turned towards empirical experience with their use in PCS projects.

4.1. Case studies

Case-studies of companies engaged in PCS projects were carried out by the author during 2004-2005. Six of the investigated cases are included in this article. The first four cases concern companies who had implemented a PCS, which was in operation. These four cases were investigated by conducting un-structured and semi-structured interviews with eleven people from these companies, along with reviews of earlier studies and analysis of documentation produced by the companies. The last two case studies concern ongoing projects, which were investigated by conducting observations of five modelling sessions followed by interviews. The modelling sessions involved model-managers and domain experts who discussed and refined their conceptual product model, which was created by using the PFMP notation and presented on printouts (the two studies are described more in detail in (Haug and Hvam, 2005a)). The interviews were given on condition of anonymity, which is the reason why the four companies having implemented a PCS are named A through D, and the two companies with ongoing projects are named E and F.

The PCS's of the companies A through D support sales processes, and each include several product families, which consist of several hundred or even thousands of different components and rules. The PCS's of company E and F are at the prototype stage and they both include only one product family with less than a hundred components, but more than a hundred rules. The PCS of company E is aimed at engineering design, while the PCS of company F supports the sales process. In the following, the use of notations in the six cases is described. Instead of describing the initial approach towards developing the PCS's of company A to D, the focus in these cases is on what they do today when updating and adding models to their PCS's, as their current approach reflects the lessons learned and therefore holds more useful information.

Company A uses PFMP's together with CRC-cards for knowledge elicitation. In the design and maintenance phase the PFMP's are simplified to only display part-of structure without attributes or kind-of structure, i.e. just a hierarchical list of product components. This is combined with extended CRC-card, as proposed by the CPM-procedure, but with additional fields for management of changes. A major reason for this simplified use of the PFMP notation is that their IT-based documentation system does not allow other model elements to be graphically displayed. The company, however, expressed interest in being able to do so. In the early days of the project the company had tried to apply class diagrams during knowledge elicitation, but it turned out to be too difficult to get most domain experts to understand this notation, for which reason the approach was abandoned except for modelling other aspects than the knowledge base. The company expressed limitations of PFMP's regarding repeated product substructures, and when creating graphical representations including combinations of product families.

Company B uses PFMP's together with CRC-cards for knowledge elicitation. After this the PFMP's are discarded and the rest of the product knowledge is modelled in extended CRC-cards where the graphical overview of the product structure is the one provided by the PCS (a hierarchical list) and their documentation system (a non-hierarchical list). The elements of their products have several combinational possibilities, wherefore attention to a greater degree is paid to modelling combinational constraints within the components rather than on diagrams that show all components. This company had experimented with the use of class diagrams together with domain experts, which showed that too much training was required, and consequently it was not considered as an alternative. On the other hand the company uses class diagrams for modelling other aspects of the PCS than product knowledge.

Company C applies PFMP's for knowledge elicitation, which is supplemented by structured documents where rules and other specifications are written. In the design phase they use a diagram type created by a vendor of commercial PCS's, which is tailored towards the specific PCS software. Both their PFMP's and the commercial diagrams are maintained, which means that most of the product knowledge in the PCS is documented twice. The argument for using the commercial diagram is the lack of model concepts supported by PFMP's. The reason why PFMP's are used is that most domain experts understand this in contrast to the commercial diagram type. A part from the fact that PFMP’s do not support certain modelling concepts, the company expressed that they found it elaborative to use PFMP's for modelling interrelated product families and when dealing with repeated component structures. The company is currently considering how to reduce the documentation work, where class

diagrams are considered as an alternative to PFMP's, as this would allow creating models with more resemblance to the commercial diagram, and hereby minimise conversion work between diagrams.

In contrast to the other companies company D does not deal with big product structures, and a typical configuration only involves relatively few components. The product knowledge is mainly delivered from product developers as lists of rules. Initially, PFMP's were used to create design models, where the part-of structure could be transferred directly to the PCS. Later, a newer version of the PCS software has been acquired, which in contrast to the older version also supports basic elements of the class diagram notation in the modelling environment. Besides this aspect, several classes of the model have multiple parents, which as mentioned is something that is better shown in class diagrams. In the current modelling work being done, basic notation from class diagrams is used.

Company E decided to develop their first two prototypes based on a revised version of the CPM-procedure (Haug and Hvam, 2005b), which are aimed at the development of PCS's that include 3D models. This revised version of the CPM-procedure proposes the use of PFMP's for knowledge acquisition, simplified PFMP's with CRC-cards for the design of the knowledge base, and class diagrams for defining user interfaces and system interfaces. Overall, the experience gained from the use of PFMP's was good in the sense that the domain experts only required minimal introduction to the notation in order to understand this. Class diagrams were used for the design of an interface application, put on top of the commercial 3D configuration software. However, the domain experts were not exposed to these models.

Company F based the development of their first prototype on the CPM-procedure. Therefore, PFMP's were used for knowledge elicitation and class diagrams with CRC-cards for the creation of the design model. The experience with PFMP's in this case was that the involved domain experts understood these with hardly any introduction. However, the analysis phase resulted in five separate PFMP models, as the model-managers found the PFMP notation inadequate for modelling the interrelations of these models. The PFMP models were therefore transferred to class diagrams where the number of classes was reduced, and the five models were connected to each other. In this case, the domain experts were not involved in the creation of the class diagram models.

4.2. Discussion of results

Several of the investigated companies expressed problems due to expressional weaknesses of the PFMP notation. These weaknesses revolve around the difficulties in expressing multiple parents, problems with interrelated models and the limited range of model elements. On the other hand, the PFMP notation was considered to have higher learnability than class diagrams, as all of the investigated companies who had knowledge about both class diagrams and PFMP's preferred PFMP's for knowledge elicitation from domain experts. Some of the investigated companies even had experience with the use of both PFMP's and class diagrams for elicitation of product knowledge, which led to the same conclusion. The observations in the two ongoing cases confirmed that it is possible for domain experts without modelling experience to understand PFMP's with minimal introduction.

Based on the experience gained from the case studies it seems that the PFMP notation has higher learnability than class diagrams - but why is that? It does not seem that the richness of the class diagrams was the main explanation for the lower learnability, as neither of the investigated companies had presented the full class diagram notation for domain experts, but a subset similar to the one of CPM-procedure. Instead, the way that the different concepts are expressed in PFMP's may explain the claims of the higher learnability than the one of class diagrams.

In a PFMP a hierarchical list is used to express whole-part relationships, compared to the diamond symbol in a class diagram. The use of hierarchical lists is a universal way of expressing that something is a part of something else, which is known from tables of contents, bills of materials etc. The meaning of this part of the model therefore requires no prior knowledge of symbols. The way generalisation is expressed in a PFMP also seems to be relatively intuitively understood, where one could expect the information in the model (i.e. names of component types) together with the separate placement of the two relationship types as being more important factors than the use of symbols. By placing generalisation and composition relationships in separate columns, these relationships can be understood based on their placement instead of the use of symbols. The constraint relation in a PFMP

is expressed by drawing a line from one element to another. Contrary to class diagrams, this is the only relation type that is expressed by drawing lines between elements, not directly succeeding each other. This makes it quite intuitive that this is some kind of non-hierarchical relation. All in all it seems that compared to class diagrams PFMP's use more commonly known ways of representing these abstraction mechanisms.

The placement of classes in PFMP's results in a reading pattern resembling the one of a text, i.e. reading from left to right, one step down and so on. For models consisting of many classes this aspect contributes to making the representation less confusing for domain experts than it would be the case with normally drawn class diagrams. The prescribed placement of classes in a PFMP also means that few considerations regarding where to place model elements are required, which for inexperienced model-managers might be a comfortable guideline. Besides making the representation easier to read, the fixed model structure also has an advantage during modelling sessions. As described the idea for the execution of the knowledge elicitation is that a PFMP goes through continuous refinements in sessions with participation of model-managers and domain experts. In these sessions the elements of the diagram are reviewed, where the natural reading pattern of PFMP's makes this revision procedure quite intuitive. Class diagrams, as they are normally drawn, do on the other hand not invite to this pointwise review of its contents.

It seems that the PFMP notation have high learnability and support modelling sessions very well because of its structure, but this is not necessary a valid argument of using PFMP's, as class diagrams easily could be elaborated with the same structure principle of a PFMP. This, however, makes the symbols for composition and generalisation superfluous. In figure 4, part of the example in figure 2 is drawn in a class diagram using the placement structure of the PFMP, where individual methods have been added to the types to indicate a presumed difference in their features.

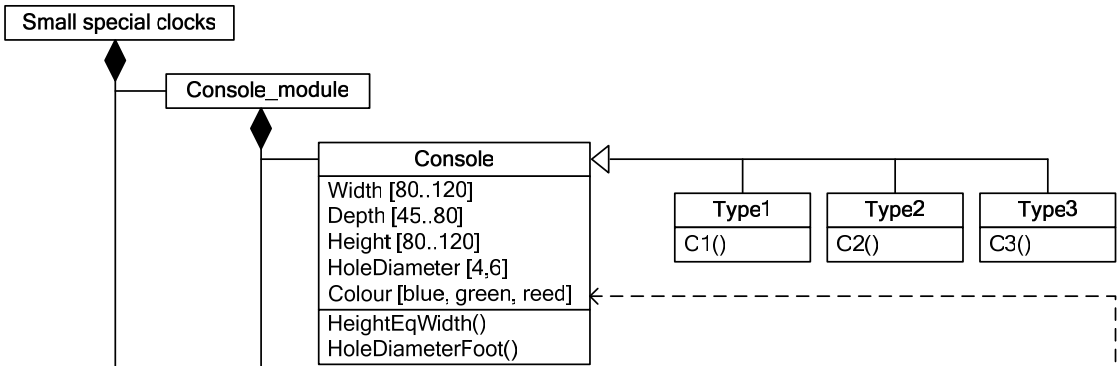


Figure 4: Product Structured Class Diagram

Using the fixed structure of PFMP's is not without trade-offs. Another downside, besides the expressional limitations this causes, is the format of the representation. For products with many components represented in a PFMP the vertical side of the model is many times longer than the horizontal, since the classes in the whole-part structure are always placed below each other. Class diagrams do not have this limitation for which reason classes to some extend can be organised accordingly to the paper format that one wishes to use.

Another problematic aspect of the ways relations are drawn in a PFMP concerns the kind-of relation. As some companies mentioned, a PFMP is not very suitable for representation of several levels of generalisation compared to class diagrams. On figure 5 an example with only to levels of generalisations are shown on respectively a PFMP and a class diagram. One could imagine the clarity of the PFMP model if additional layers of inheritance were added.

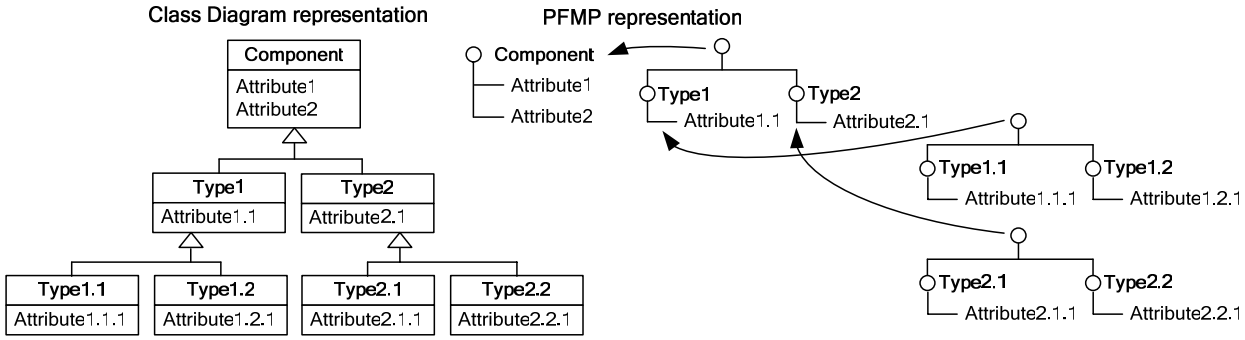


Figure 5: Multiple levels of generalisation

Besides the mentioned aspects (expressional strength and appearance of the representation) several other aspects also must be taken into consideration when comparing the two types of diagrams. The level of formalism and instructions in the use of PFMP's are very sparse compared to class diagrams and had in the studied cases resulted in different kinds of interpretations of the concepts. The low formalism was by some companies commented as leading to some confusion regarding issues as: when and how to use the kind-of-relation oppose to attributes, which constraints to represent in a constraint relation, which classes these constraints later should belong to, and how to deal with repeated components. On other hand, all the concepts of class diagrams are well defined by OMG (2005).

The range of symbols for representing different concepts in class diagrams is much greater than the one of PFMP's. It was observed in several of the cases that the limited variety of concepts of PFMP's had led to new notation elements being invented for expressing other concepts. It is obviously an advantage to be able to adapt a notation on-site to a specific modelling challenge, but doing this always produces a risk of misinterpretation from other users of the representation. On the other hand, class diagrams have wide selection of predefined concepts that can be used.

The possibilities for IT-supported modelling are also worth considering when choosing a modelling language. Two of the interviewed companies had documentation management systems, which they had adapted for administrating CRC-cards. None of these systems supported the full PFMP notation (only hierarchical lists of components) wherefore other tools were used when creating PFMP's in the knowledge elicitation phase. Not having a modelling tool that fully supports the PFMP notation makes construction and maintenance of the diagrams time-consuming, and means that notational errors will occur more frequent since there is no automatic control of if the notation is used correctly. On the other hand, several modelling tools that support modelling in UML exist.

One should keep in mind that modelling takes place at different stages of a project. So if the PFMP notation is used for knowledge elicitation and this model is later transformed and further developed in class diagrams (as prescribed by the CPM-procedure), either the class diagrams are the only current documentation or the PFMP models have to be updated. If there is little need for domain experts to review already built models, moving from PFMP to class diagrams might be a good approach, but if domain experts have to review documentation created with class diagrams, it might be better to use class diagrams from the start, since training in the class diagram notation is required anyway.

The decision of which notation to use should also be influenced by the PCS chosen. If the PCS does not support certain concepts, there is no point in modelling them this way. Many PCS's display structure in hierarchical lists, which means that using PFMP's could facilitate the transition from conceptual to implemented model better than class diagrams.

5. CONCLUSIONS

In this article the PFMP and class diagram notation were compared in respect to their applicability for modelling knowledge bases of PCS's. The comparison was executed in the two main dimensions, expressional strength and usability. Six case studies were used as a basis for these investigations.

The comparison of the expressional strength showed several limitations of PFMP's compared to class diagrams, whereas no advantages appeared in this respect. The weaknesses of PFMP's concerned the limited range of modelling concepts, that it does not support modelling of multiple parents very well, and that it is inflexible when it comes to creating interrelated models. On the other hand, it seems that the PFMP notation is easier to learn than class diagrams for persons without modelling experience, which was the opinion of all the studied companies that knew of both notations. This phenomenon might be explained by two main characteristics of PFMP's. Firstly, the way the relationship types of a PFMP are represented to some degree resemble well-known concepts, contrary to the symbols of class diagrams. Secondly, in a PFMP different relationship types are placed in different columns, where they on normally drawn class diagrams are mixed together. The fixed placement of classes in PFMP's implies a reading pattern, which resembles the one of text, contrary to normally drawn class diagrams. Due to this aspect, the PFMP structure seems to be very well-suited when reviewing existing, maybe only partly constructed, models. The limitations of the usability of PFMP's emerge when more complex concepts are modelled. For instance, it was shown that the PFMP notation is not well-suited for modelling multiple levels of generalisation.

Other aspects must be also taken into consideration when choosing notations in a PSC project. One should be aware of the low formalism of the PFMP, which in the investigated cases had resulted in different non-normative extensions of the language, which represents a risk of misinterpretation. Furthermore, no real IT-based documentation tools currently support the entire PFMP notation, for which reason programs with no automated modelling support, like MS Visio and Excel, must be used. The question of who is to maintain which models should also be considered, as starting with a simple notation, which later has to be replaced with a more complex one, means that domain experts will have to learn the complex notation anyway if they are to use the models. Also the choice of PCS should affect the choice of notation, as if concepts cannot be implemented in the PCS, there is no point in defining them this way.

Because the learnability of the PFMP notation to a great extent can be explained by the fixed placement of its model elements, this structure was applied to class diagrams. As such *product structured class diagrams* might have usability similar to the one of PFMP's and as these can be created without violating normative UML it seems that the arguments for choosing the PFMP notation are very few. The benefits of choosing PFMP's in this light boil down to preferring other symbols for representing different concepts. But applying the fixed structure from PFMP's to class diagrams does not directly solve the issues concerning multiple levels of generalisation and being able to display multiple wholes, as class diagrams inherit these limitations by doing so. Nevertheless this might in some cases turn out to be a useful compromise.

All in all, this article has provided a better basis for choosing between graphical notations to be used when developing PCS's by clarifying the strengths and weaknesses of class diagrams compared to PFMP's, and by discussing other aspects to consider when making this choice.

6. REFERENCES

1. Beck, K., and Cunningham, W.A. (1989). A laboratory for teaching object-oriented thinking. Proceedings of OOPSLA'89 and Special issue of SIGPLAN Notices, 24(10): 1-6.
2. Felfernig, A., Friedrich, G., and Jannach, D. (2000). UML as domain specific language for the construction of knowledge based configurations systems. International Journal on Software Engineering and Knowledge Engineering, 10(4): 449-470.
3. Felfernig, A., Friedrich, G., and, Jannach, D. (2001). Conceptual modeling for configuration of mass-customizable products. Artificial Intelligence in Engineering, 15(2): 165-176.
4. Fowler, M. (2005). UML Distilled Third Edition: A Brief Guide to the Standard Object Modeling Language, Addison-Wesley, Boston MA.
5. Grudin, J. (1992). Utility and Usability: Research Issues and Development Contexts. Interacting with computers, 4(2): 209-217.
6. Guéhéneuc, Y.-G., and Albin-Amiot, H. (2004). Recovering Binary Class Relationships: Putting Icing on the UML Cake. Proceedings OOPSLA'04, Vancouver, BC, Canada, pp. 301-314.

7. Haug, A., and Hvam, L. (2005a). Product analysis as a basis for building product configuration systems, Proceedings of MCPC2005, Hong Kong University of Science and Technology.
8. Haug, A., and Hvam, L. (2005b). Developing 3D Configuration Systems for manufacturers of complex building components. Proceedings of IMCM05, Klagenfurt, Austria.
9. Holzinger, A. (2005). Usability Engineering Methods for software developers. Communications of the ACM, 48(1): 71-74.
10. Hvam, L. (1994). Application of product modelling – seen from a work preparation viewpoint. PhD thesis, Department of Industrial Management and Engineering, Technical University of Denmark, Lyngby, Denmark.
11. Hvam, L. (1999). A procedure for building product models. Robotics and Computer-integrated Manufacturing, 15(1): 77-87.
12. Hvam, L. (2004). A multi-perspective approach for the design of Product Configuration Systems - an evaluation of industry applications. International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems, Technical University of Denmark.
13. Hvam, L., Riis, J. (2003). CRC-Cards for Product Modelling. Computers in Industry, 50(1): 57-70.
14. Hvam, L., Riis, J., and Malis, M. (2002). A multi-perspective approach for the design of configuration systems. Proceedings of the 15th European Conference on Artificial Intelligence, Lyon, France.
15. Mortensen, N.H. (1999). Design modelling in a Designer's Workbench – contribution to a Design Language, PhD thesis, Department of Control and Engineering Design, Technical University of Denmark, Lyngby, Denmark.
16. Mortensen, N.H. (2001). Improving business Conceptual Modeling. Invensys CRM, Copenhagen.
17. Mortensen, N.H., Yu, B., Skovgaard, H., and Harlou, U. (2000). Conceptual modeling of product families in configuration projects. Workshop at the 14th European Conference on Artificial Intelligence, Berlin, Germany.
18. Nielsen, J. (1992). The usability engineering life cycle. Computer, 25(3): 12-22.
19. Nielsen, J. (1993). Usability Engineering. Boston Academic press, Boston, MA.
20. Odell, J., Parunak, H.V.D., and Bauer, B. (2000). Extending UML for agents. Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence, Austin, Texas.
21. OMG. (2005). Unified Modeling Language: Superstructure: Version 2.0, formal/05-07-04. Nedham, MA.
22. Pine, B.J., Victor, B., and Boynton, A.C. (1993). Making Mass Customization Work. Harvard Business Review, 71(5): 108-119.
23. Priestley, M. (2003). Practical Object-Oriented Design with UML. 2nd edition, McGraw-Hill, New York.
24. Pulkkinen, A. (2000). A framework for supporting development of configurable product families. Proceedings Norddesign (Ed. Hansen, C.T.), Technical University of Denmark, Lyngby, Denmark.
25. Sabin, D., and Weigel, R. (1998). Product Configuration Frameworks - A survey. IEEE Intelligent Systems & Their Applications, 13(4): 42-49.

Paper B3

Product Structured Class Diagrams to support the development of Product Configuration Systems

Haug, A. and Hvam, L. (2007): "Product Structured Class Diagrams to support the development of Product Configuration Systems", in Proceedings of MCPC 2007, Boston, Oct. 7-10, 2007.

Product Structured Class Diagrams to support the development of Product Configuration Systems

Anders Haug and Lars Hvam

Abstract

For several companies the use of product configuration systems (PCSs) has produced a range of benefits such as minimising the use of resources and shortening the lead times in product specification processes. When developing a PCS, two kinds of models are often created, namely analysis and design models. The task of describing domain knowledge in analysis models often involves domain experts, for which reason the analysis language has to be easily understandable in order to avoid extensive training. For this task the so-called Product Variant Master (PVM) diagramming technique is often applied. On the other hand, the creation of design models does not involve domain experts to the same degree, and the requirements for the design language are more focused on having a formalised and rich language. For this task class diagrams are often applied. To avoid the use of different modelling languages in the analysis and design phase, this paper proposes the use of a layout principle, named Product Structured Class Diagrams (PSCDs), which incorporates the usability of PVMs into the class diagram notation. To support this proposition, it is investigated if PSCDs hold the qualities of both PVMs and class diagrams.

Keywords: Product configuration, knowledge engineering, object-oriented modelling

1. Introduction

A product configuration system (PCS) can be defined as a product-oriented expert system (or knowledge-based system) which allows users to specify products by selecting components and properties under restriction of valid combinations. For several companies the use of PCSs has produced a range of benefits, such as: shorter lead times (sales to delivery), improved quality of product specifications, preservation of knowledge, the use of fewer resources for specifying products, optimized products, less routine work, improved certainty of delivery, and less time needed for training new employees (Hvam, 2004; Hvam et al., 2007; Riis, 2003; Forza and F. Salvador, 2002; Forza et al., 2005).

The development of a PCS implies that domain expert knowledge is elicited and later represented in the PCS. The representation of domain knowledge is often one of the greatest tasks in a PCS project (Sabin and Weigel, 1998; Hansen et al., 2003; Edwards and Ladeby, 2005). To facilitate the transfer of knowledge from domain experts to the knowledge base of a PCS, two distinctive kinds of models are often applied, namely *analysis models* and *design models*.

Analysis models (also called domain or conceptual models) are used for describing a real world domain of interest before initiation of design and coding (Priestley, 2003). In PCS projects, analysis models can be seen as mediators between product technical domain experts and knowledge engineers, in the sense that analysis models can be used for capturing what the domain experts articulate and for letting the domain experts evaluate if the captured knowledge is correct. Obviously, the domain expert must understand the representation language in order to be able to evaluate the analysis models. If the creation of analysis models involves domain experts who are not familiar with conceptual modelling, the applied modelling technique has to be easy to learn in order to avoid extensive training.

The model that specifies what should be implemented in the PCS software is called a design model. Design models are typically created by persons who are familiar with the software in which the models are to be implemented and who posses modelling expertise. The requirements for a design language are therefore less about how learnable it is and more about how well it allows expressing what should be implemented in a PCS in an adequately exact and formalised manner.

Having different requirements for the modelling techniques that are used to create analysis and design models, the question is whether to apply the same technique or two specialised techniques for the two tasks. The use of two different modelling techniques means that there is a need for a transfer of information from analysis model to design model, which can be a time-consuming task and a possible source of errors. On the other hand, the use of the same modelling technique for the creation of the two kinds of models could mean that this either requires significant training of domain experts or that it becomes troublesome to express the design in a complete and unambiguous manner.

In PCS research that deals with diagrammatic representation languages two types of diagrams are often mentioned, namely product variant masters (PVM) and class diagrams. While class diagrams are richer, more flexible and more formalised than PVMs, PVMs are by some researchers considered to be more easily understandable than class diagrams, why their approaches prescribe PVMs for the elicitation of domain expert knowledge. In this paper a diagrammatic principle that aims to include the advantages of both PVMs and class diagrams is investigated and further elaborated on.

The remainder of this paper is structured as follows. In section 2, the strengths and weaknesses of PVMs and class diagrams are discussed, which is followed by an analysis of the different approaches for developing PCSs that include these techniques. Next, in section 3, the principle of product structured class diagrams (PSCDs) is presented. In section 4 and 5 the utility and usability of PSCDs are investigated. The paper ends with a conclusion in section 6.

2. Approaches for the development of product configuration systems

Three different kinds of approaches for how to apply diagrammatic models for the analysis and design of PCS knowledge bases have been identified in the literature. The first approach prescribes the use of PVMs for the creation of both analysis and design models, the second approach prescribes the use of class diagrams for both the creation of analysis and design model, while the third approach prescribes the use of PVMs for the creation of analysis models and class diagrams for the creation of design models. Before analysing these different approaches, PVMs and class diagrams are resumed and compared.

2.1. PVMs

The PVM technique (also referred to as a product family master plan) provides a formal way of representing a product assortment (Mortensen et al., 2000, Hvam et al., 2007). The PVM technique is far less widespread in use than class diagrams, and compared to class diagrams it is more a product description technique than a software modelling technique. A PVM consists of two generic sections, *part-of structure* and *kind-of structure*. The part-of structure defines the parts of which a given product family can consist, while the kind-of structure describes the variation of a part, i.e. different types with common characteristics. To the left in figure 1 the notation formalism of the PVM technique as it has been defined until recently is shown, and to the right the latest definition of the formalism is shown.

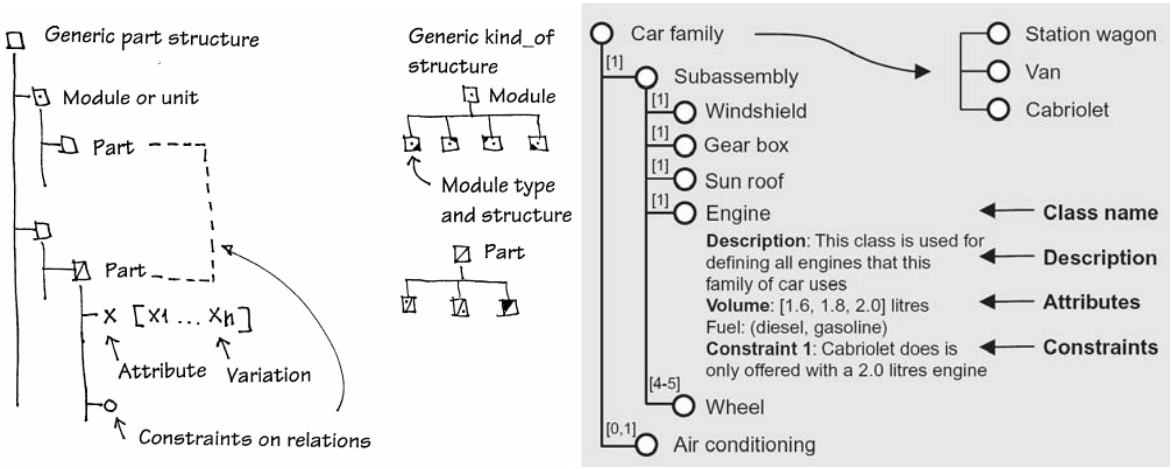


Figure 1: PVM formalisms (left: Mortensen et al., 2000, right: Hvam et al., 2007; Harlou, 2006)

The main changes of the PVM definition are: the term *class* is used instead of *module*, *unit* and *part*; kind-of classes are lined up on a vertical line, while previously they were lined up horizontally; constraints are no longer shown by drawing lines between classes, but are written below a class; cardinality is shown; and descriptions of classes are included in the formalism.

2.2. Class diagrams

Class diagrams are part of the Unified Modelling Language (UML). UML was officially adopted as a standard by the Object Management Group (OMG) in 1997 and have widespread use within software development. UML 2.0 includes thirteen types of diagrams, which are divided into three categories: six s*tructure diagrams* (including class diagrams), three *behaviour diagrams* and four *interaction diagrams* (subtype of *behaviour diagram*s) (OMG, 2005).

The class diagram describes object classes and their relations, and is the most commonly applied UML diagram (Fowler, 2005). The notation for the class element and the most common relationship types are shown in figure 2. Furthermore, a *navigability arrow* can be used to show the direction of association, aggregation and composition relationships.

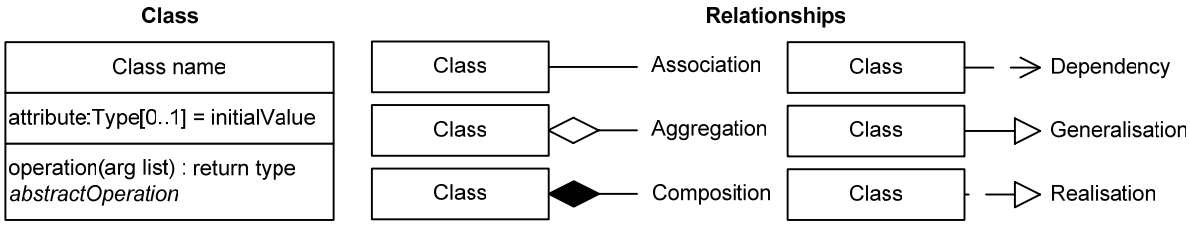


Figure 2: Elements of class diagrams

Besides what is shown in figure 2, expressions of constraints regarding how elements can be combined are essential in a PCS context. UML includes a formal constraint language called OCL (Object Constraint Language). However, the use of OCL is not required to obey normative UML, since any formalism used for describing constraints is allowed as long as placed within braces ({}) (Fowler, 2005). A constraint string may be placed inside or near the model element that it describes, e.g. after an attribute or next to a relation between two classes. Constraint strings may also be placed in the UML note symbol.

Another important concept in UML is stereotypes. A stereotype represents a variation of an existing type of model element (e.g. a class or a relation) and thereby enables the use of platform or domain specific terminology or notation. Any UML model element can be extended by a stereotype (OMG, 2005), and the name of a stereotype is placed in guillemets (<<...>>). Stereotypes can, together with tagged values and constraints, be collected in profiles, which are packages that contain customised model elements for a specific purpose (OMG, 2005).

2.3. PVMs compared to class diagrams

Some literature considers PVMs to be easier to understand, while class diagrams provide a richer, more flexible and formalised notation (e.g. Hvam, 2004, Hvam et al., 2007). This assumption was investigated in an article where the expressional strength and usability of the two diagram types were compared (Haug and Hvam, 2006). The article points out that class diagrams seem to have the following advantages compared to PVMs: 1) A more unambiguous and well-defined notation formalism; 2) Widespread use within software development; 3) The possibility of modelling other aspects than structure within the same language (i.e. UML); 4) Much standard software support the elaboration of class diagrams, while the PVM notation is not supported by any standard software; 5) More predefined relationship types than in the PVM notation; 6) A means of creating new types of model elements within normative use of the notation; 7) A larger range of predefined model elements to symbolise various concepts; 8) Notation for modelling relations between different models; and 9) Allow the display of classes, which belong to multiple wholes or inherit from more than one class, without having to show the same class more than once in the diagram. Besides these advantages, Haug and Hvam (2006) also mention that class diagrams allow a more flexible paper format, since model elements can be placed according to a preferred height-width ratio, while PVMs tend to get very long and slim. However, the small practical inconvenience related to the paper format when using PVMs is outweighed by other benefits, as it will be explained later in this paper. On the other hand, the use of PVMs seems to have two major advantages, namely the learnability of the notation (i.e. how fast users can learn and start using the technique (Nielsen, 1992)), and that it is well-suited for stepwise revision of its content. The looser formalism of the PVMs, which above was mentioned as a weakness compared to class diagrams, might in some contexts be considered as a strength in the sense that PVMs are therefore easier to use. However, class diagrams might just as well be used without paying attention to the defined formalism, why this cannot be considered a real advantage. A similar claim could be that the use of PVMs in a domain analysis situation helps to avoid the use of UML terminology, which would not be comprehensible to many of the product technical domain experts who are involved in a PCS project. However, for skilled knowledge engineers it should not be a problem to avoid this.

The learnability aspect of PVMs compared to class diagrams was investigated by studying PCS projects at six companies (Haug and Hvam, 2006). The study showed that all the investigated companies considered the learnability of PVMs to be higher than the learnability of class diagrams. Two of the companies had even tried to apply class diagrams for the creation of models together with domain experts, but had abandoned this approach and instead chosen to use PVMs, since these seemed to be easier for the domain experts to understand. Although most of the investigated companies had chosen to apply PVMs, several of them expressed that they had faced problems due to the expressional limitations of the technique.

To illustrate the use of PVMs and class diagrams, a basis is taken in a simple clock construction, illustrated in figure 3. The same example of a generic model of such a clock construction is shown in figure 4 and 5 by using PVMs and class diagrams. The way the PVM is represented in figure 4 differs from the defined notation of Hvam et al. (2007), but includes some elements that are often applied in practice, according to the experience of the author (Haug and Hvam, 2006). The differences are that

the PVM-model in figure 4 includes constraint relations, constraint references, and a sheet for expressions of constraints, while the class description is left out.

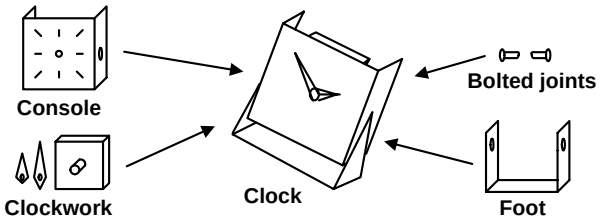


Figure 3: Simple clock construction

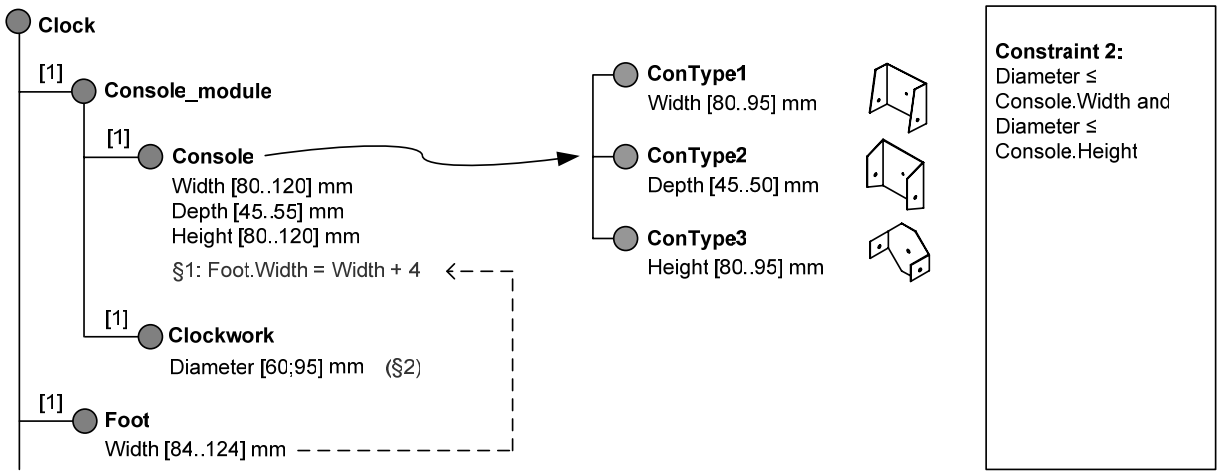


Figure 4: Example of the PVM notation

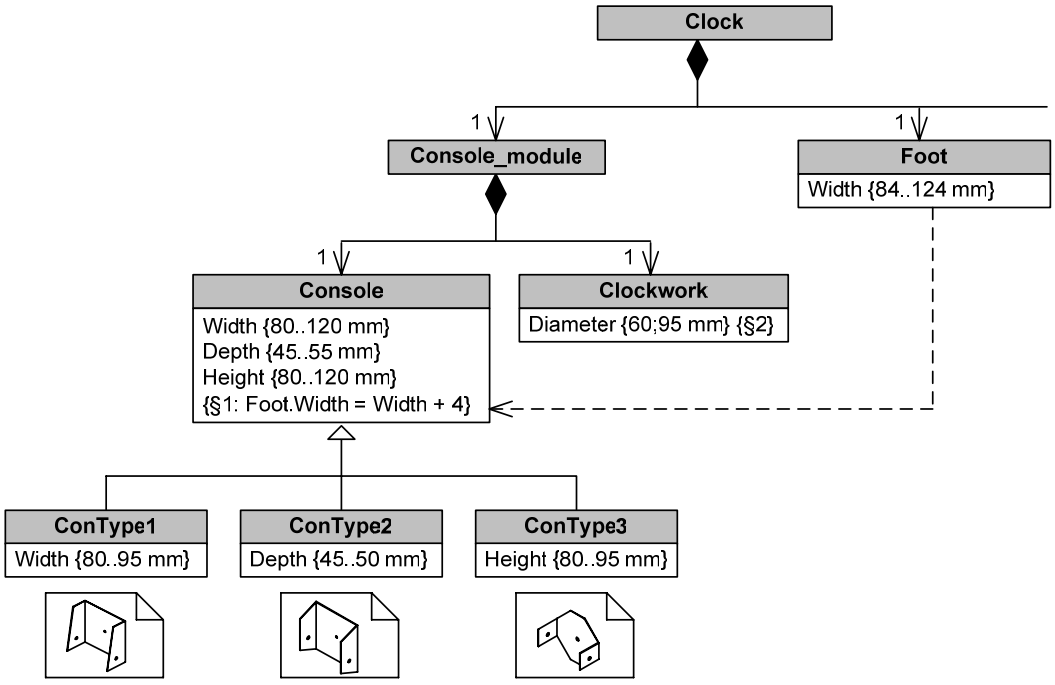


Figure 5: Example of the class diagram notation

Due to the limited size of the models in figure 4 and 5, it might not be clear why the use of the PVM notation is considered to produce more easily comprehensible representations than the use of class diagrams. But the PVM technique has some characteristics that could explain why the studies by Haug and Hvam (2006) indicate that the PVM technique has better usability in modelling sessions of configuration projects. The high usability of PVMs compared to class diagrams can be explained by two main characteristics. Firstly, in that the relationship types of a PVM are placed in separate columns, while relations in a class diagram are mixed without any predefined rules of placement. This means that by using PVMs instead of class diagrams it is easier to read parts of the model separately and the reading pattern resembles text, which can be a great advantage during sessions where the contents of a model are reviewed stepwise. The second characteristic of PVMs that may explain their usability compared to class diagrams is that PVMs include more easily understandable symbols for its three relationship types. Part-of/composition in a PVM is shown by using a hierarchical list, which is a well-known way of expressing whole-part structure, compared to the diamond symbol of class diagrams. The constraint relation of a PVM is drawn as a dotted line between classes which do not necessarily directly succeed each other. This makes this relationship easy to recognise, since this is the only relationship in a PVM with this characteristic, compared to class diagrams where all the relationships are drawn in the same way, but by using different symbols on the lines and different types of lines. On the other hand, the way kind-of relationships are represented in PVMs does not differ much from how generalization is shown in a class diagram.

2.4. Approaches for the developments of PCSs

As mentioned, three different kinds of use of PVMs and class diagrams in approaches for the development of PCSs have been identified in the literature.

Mortensen et al. (2000) propose a five-phase procedure for conceptual modelling of product families in configuration projects. In this procedure the prescribed means for describing the structure of a product assortment is the PVM technique. Later each class of a PVM is further specified in so-called *Class Description cards* (CD-cards), followed by *Class Responsibility cards* (CR-cards). Compared to the original CRC-cards (Beck and Cunningham, 1989), which only consist of the class name together with two columns for responsibilities and collaborators, CD/CR-cards include a range of additional fields. Since PVMs are the only prescribed structural diagram and that PVMs could be modified during design, PVMs can thereby also be said to be the part of the design language of this procedure.

An approach for developing PCSs, which prescribes the use of class diagrams for developing PCSs, has been proposed by Felfernig et al. (2000; 2001). They base their choice of UML on arguments of the wide appliance in industrial software development and their good experience with the use of UML designs for validation by technical domain experts. The idea of the approach is firstly to extend the UML model by commonly used configuration concepts, and secondly to create a definition of how these concepts are mapped to a configuration language (Felfernig et al., 2000). The development process by Felfernig et al. (2000; 2001) prescribes the use of class diagrams for the creation of analysis models. The analysis models are after checks transformed into the knowledge base, and can later be modified during tests if unexpected results of the implemented knowledge should emerge. Class diagrams, therefore, in principle represent both analysis and design models.

An approach for the development of PCSs, which includes the use of both PVMs and class diagrams, is a procedure from the Centre for Product Modelling (CPM) at the Technical University of Denmark. The CPM-procedure was introduced in 1994 (Hvam, 1994) and has since undergone changes as experience with the procedure has been acquired (Hvam et al., 2007). In its current form, the CPM-procedure includes seven phases to support the development, implementation and maintenance of PCSs. The CPM-procedure aims to capitalise from the use of both PVMs and class diagrams in the same development process by benefiting from the usability of PVMs for the elicitation of product knowledge from domain experts, while using the richer, more formalised and more flexible class diagrams for the design of the knowledge base of a PCS. Originally PVMs were not included in the procedure, but since early experience showed that domain experts often had difficulties in understanding the notation of class diagrams, the somewhat simpler PVM technique was included. To allow detailed descriptions of the classes while avoiding having too much information in a class diagram (or PVM), the CPM-procedure proposes the use of special CRC-cards, which, compared to

the original CRC-cards (Beck and Cunningham, 1989), include several additional fields (Hvam et al., 2003, Hvam et al., 2007). Hvam et al. (2007) recognise that in some projects the creation of PVMs with matching CRC-cards would provide an adequate basis for the creation of the knowledge base of a PCS, for which reason class diagrams should not be elaborated.

2.5. Analysis of approaches for the developments of PCSs

Presuming that the claim that PVMs have better usability than class diagrams is correct, the knowledge elicitation process could become much more unproblematic by applying PVMs instead of class diagrams. However, if PVMs are applied for both analysis and design models, it may not be possible to create adequately exact and formalised design models due to the limited range of model elements and the loose notation formalism. The objective of the CPM-procedure of capitalising from the advantages of PVMs in the knowledge elicitation phase and the advantages of class diagrams in the design phase therefore seems to make good sense. The CPM approach, however, requires that a transfer of information from analysis to design model is carried out. This presents at least four problematic aspects. Firstly, the use of two different notations requires a transfer task, which is time-consuming and in itself does not add new information. However, using the same diagram type for both analysis and design could be even more time-consuming if the chosen diagram requires either much introduction or leads to difficulties in expressing a design. Secondly, a manual transfer of information from one model to another, created by using another notation, obviously includes the risk of errors being made. This can be particularly problematic when parts of a PVM-model are transferred to a class diagram while optimising the model structure, since it can then be hard to distinguish between errors and intended modifications of the model. Thirdly, there is the problem of mapping between PVMs and class diagrams. Several authors from different research areas within information systems have dealt with the problem of different languages that do not have a natural mapping, which is often referred to as impedance mismatch (e.g. Everman and Wand, 2005; Kolp et al., 2002; Cilia et al., 2003; Rozen and Shasha, 1989). This could also be the case when moving from a PVM to a class diagram, because the limited range of model elements in the PVM notation can force the knowledge engineers to invent new symbolism in order to express relevant aspects. As this may not be transferable to class diagrams in an unambiguous fashion, a mismatch can occur. Fourthly, the use of PVMs during the product analysis phase and class diagrams for design models presents a problem if the maintenance of models requires the involvement of domain experts. As the argument for using PVMs must be that they are understood by domain experts, it must also be assumed that the domain experts are only familiar with PVMs, and not with class diagrams. But as design models normally are the only updated documentation, the maintenance task requires that the domain experts to be involved must also understand class diagrams.

Based on the mentioned downsides of all three kinds of approaches, it seems that the ideal situation would be to use a diagram type that holds the advantages of both PVMs and class diagrams. A diagram type that might satisfy this requirement is described in the following chapter.

3. Introducing Product Structured Class Diagrams (PSCD)

3.1. Taking the best from the two diagram types

To avoid the transfer of information from PVMs to class diagrams, while keeping the advantages of applying both the techniques, Haug and Hvam (2006) suggest that class diagrams are to be applied for both analysis and design, but with a placement of the model elements in a way similar to the placement in PVMs. The proposed diagrammatic principle was, however, not investigated profoundly, as it was not the purpose of the article in question. The principle was named PSCDs. In figure 6 the example from figure 4 and 5 is shown using the PSCD principle.

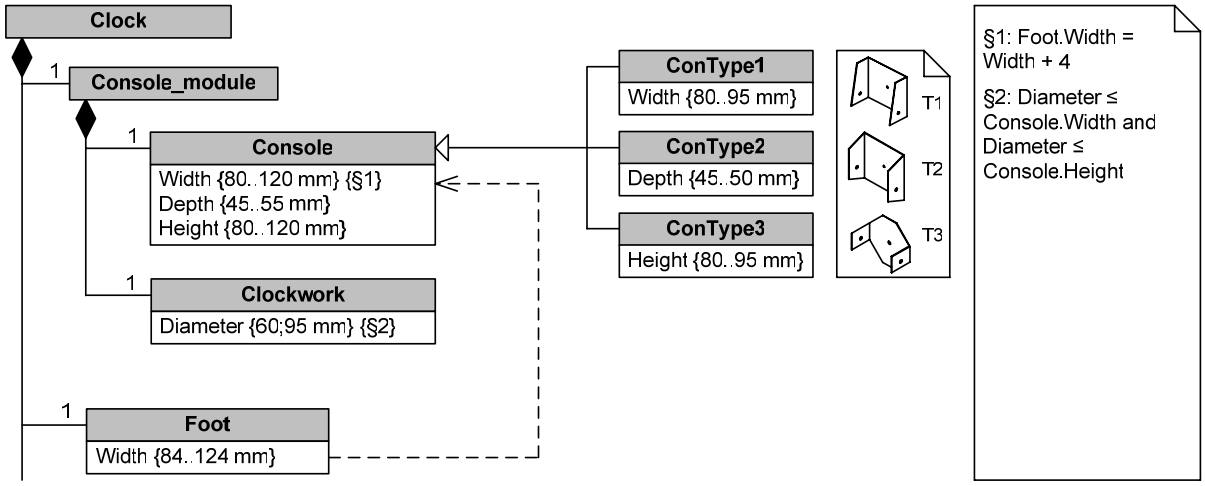


Figure 6: Example of the product structured class diagram notation

In figure 6 the two constraints are shown by placing the constraint references behind the relevant attributes and by expressing the constraints in a note. But two other ways of expressing constraints can be used, as illustrated in figure 7. The first alternative is to formulate the constraint expression below the attributes, while the second alternative is to place the constraint in a note without using references to constraints.

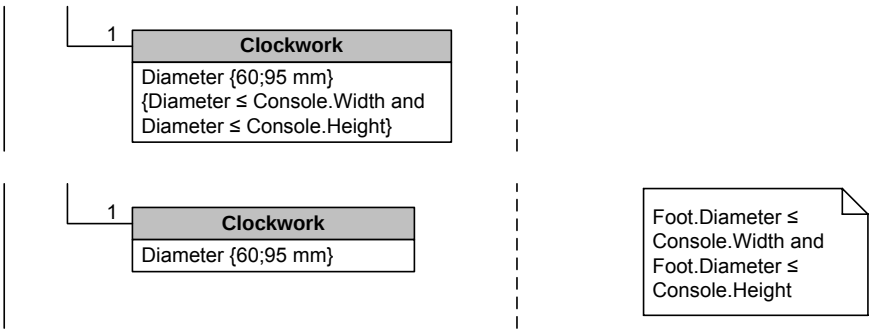


Figure 7: Alternative ways of expressing constraints

3.2. Investigating the proposed concept

For the investigation of whether PSCDs hold the same strengths as both PVMs and class diagrams, a basic distinction between *utility* and *usability* is used in this paper (Nielsen, 1992; Grudin, 1992). Utility concerns how well the design suits the (functional) needs of the user, and usability concerns how easy an interface is to use. Nielsen (1992) describes the five essential usability characteristics: learnability (how rapid users can be working with the system), efficiency (enabling users, who have learned to use the system, to attain high productivity), memorability (allowing casual users to return to the system after periods of non-use), errors (errors made by users and ability to recover) and satisfaction (how pleasant the system is to use). In the following two chapters the utility and usability of PSCDs are compared to PVMs and class diagrams.

4. Utility analysis

As mentioned, the PSCD notation was proposed as a possible solution to eliminate the problems arising from the application of different diagram types for the creation of analysis and design models, while maintaining the advantages of using both of these diagrams in a development process. To clarify whether PSCDs holds the advantages of both class diagrams and PVMs, it is firstly investigated whether the PSCD notation holds the nine advantages that class diagrams seem to have over PVMs, as described is section 2.3.

1) A more unambiguous and well-defined notation formalism: The PVM technique only includes few defined modelling elements, implying that new individual extensions often are created in configuration projects (Haug and Hvam, 2006). The class diagram notation, on the other hand, is much richer, for which reason predefined formalisms would normally be adequate. The possibility of applying well-defined and unambiguous formalisms can be advantageous when information has to be communicated through the use of diagrams, e.g. if knowledge engineers pass on models to programmers who are to implement the model. Since PSCDs use the class diagram notation formalism, PSCDs hold this advantage over PVMs.

2) Widespread use within software development: The widespread use of UML class diagrams compared to the limited use of PVMs could in some contexts make class diagrams better suited for communication. For instance, if models are passed on from knowledge engineers to programmers, who due to the more widespread use of class diagrams are more likely to understand exactly what is meant in the models. As PSCDs use the UML notation, PSCDs holds this advantage over PVMs.

3) The possibility of modelling other aspects than product structure within the same language: PVMs are not part of a bigger modelling language, while class diagrams are a part of UML. As mentioned in section 2.2 UML 2.0 includes a range of different diagram types, which means that well-defined concepts and practices for applying class diagrams together with diagrams that describe other than structural aspects can be utilised when using class diagrams. As PSCDs use the UML notation this gives PSCDs the advantage over PVMs.

4) Much standard software support the elaboration of class diagrams, while PVM notation is not supported by any standard software: Class diagrams have an advantage over PVMs when it comes to software-supported elaboration of models. Since commercial software aimed at PVMs does not exist, software like MS Visio and Excel are normally applied. On the other hand, several kinds of software support the elaboration of class diagrams, both diagramming tools and CASE-tools (see e.g. www.omg.org for a list of tools). As software for supporting the elaboration of class diagrams does not prohibit the application of the fixed structure of the model elements in a PSCD, PSCDs have an advantage over PVMs on this point.

5) More predefined relationship types than in the PVM notation: Class diagrams include a range of predefined relationship types, which do not exist in the PVM notation, as seen when comparing the PVM and class diagram formalisms in figure 1 and 2. When applying PSCDs all the class diagram relationships are part of the notation, which is why PSCDs hold this advantage over PVMs.

6) Means for the creation of new types of model elements within normative use of the notation: Stereotypes have been applied in different contexts in order to adapt class diagrams to a configuration domain. The mentioned approach by Felfernig et al. (2001) defines the following class stereotypes: *component*, *resource*, *function* and *port*; the association relationship stereotypes: *incompatible* and *is_connected*; and the stereotypes for the dependency relationship: *requires*, *produces* and *consumes*. The use of stereotypes is also found in Riis (2003), who based on the CPM-procedure defines a list of stereotypes that are found useful when creating product knowledge models. This includes *product-family*, *part*, *organ*, *function*, *property*, *life-phase system*, *factory-model* and other types of life-phase models. Although it to some extent would be possible to apply the concept of stereotypes in a PVM, this is not defined as part of the PVM notation and can therefore be perceived as an advantage of PSCDs over PVMs. In figure 8, the stereotyped associations *incompatible* and *requires*, as defined by Felfernig et al. (2000), are used to illustrate that *ConType1* cannot be selected together with *FootType3*, and that if *ConType3* is chosen, then *FootType1* must be selected. Due to lack of space, the note box for expressing constraints is not shown in the example, but, obviously, expressions of the constraints (§1 and §2) should be found in a note box or in the matching CRC-cards.

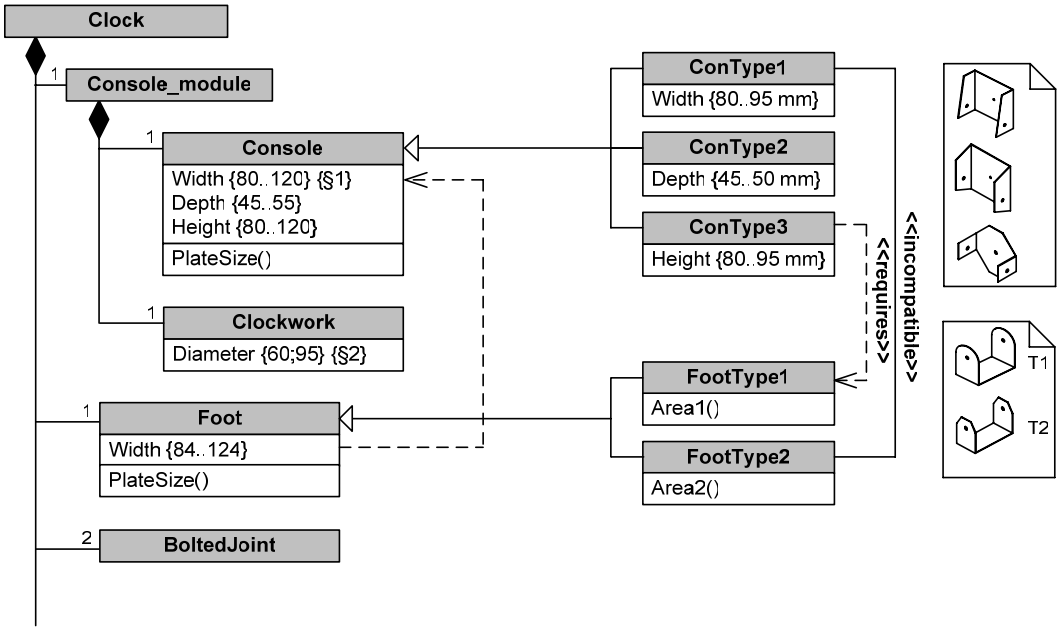


Figure 8: Application of advanced UML concepts

7) A larger range of predefined model elements to symbolise various concepts: Männistö et al. (2001) present concepts for configuration knowledge modelling by means of slightly extended UML (compared to Fowler, 1997), which includes a black square to symbolise a port for representing an interface. The need for stereotypes or non-normative UML has decreased since then as new model elements have been introduced in UML. In UML 2.0 a port element is included, symbolised by a white square. A port specifies a point of interaction between a class and its environment, where a semantically cohesive set of provided and required interfaces can be grouped by using a port (Arlow and Neustadt, 2005). For the modelling of interfaces the ball-and-socket notation was introduced in UML 2.0 (OMG, 2005). The ball icon (often referred to as lollipops) also existed in earlier versions of UML, but as the socket icon did not, dependency relations were used to show that an interface provided was required by another class. In figure 9, an example of the ball-and-socket notation is shown in a PSCD. Here, a digital clock requires power, provided by a battery. Alternatively, the example could have been elaborated with the port element placed between the classes and the interface elements.

If interfaces or ports must be shown in a PVM according to the current definitions (Hvam et al., 2007), then classes or attributes must be used for this purpose. This alternative is rather limited compared to the possibilities provided by UML, which is why PSCDs can be said to have an advantage over PVMs when modelling this kind of information.

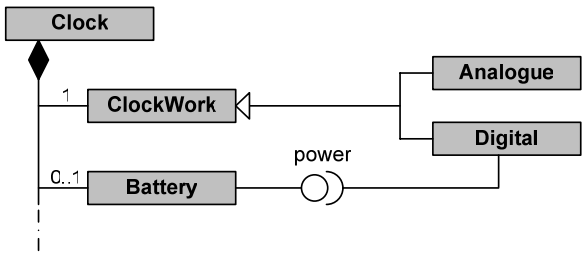


Figure 9: UML interface notation

8) Notation for modelling relations between different models: When having different PVM models, there is no prescribed graphical notation for modelling interrelationships between these. On the other hand, UML provides notation for modelling relations between different models, i.e. package

diagrams. A package is a grouping construct for organising model elements, diagrams and other packages into higher-level units. An example of the use of the package notation is shown in figure 10.

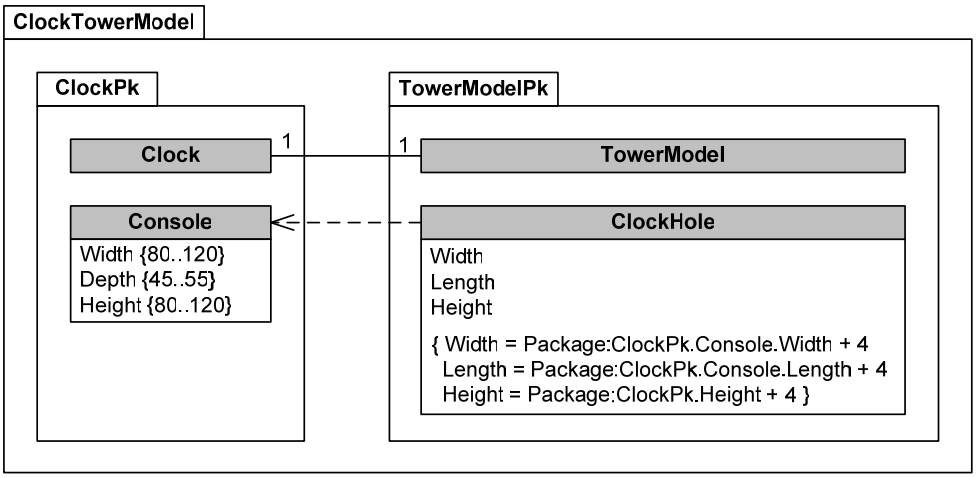


Figure 10: Package model

In figure 10 it is assumed that the model from the example in figure 6 (without the *Foot* class) is placed in a package called *ClockPk*. The clock is to be built into a miniature model of a tower, which is described in a class diagram placed in a package called *TowerModelPk.* Among other classes the *TowerModelPk* includes the classes *TowerModel* and *ClockHole*. To ensure that a clock will fit the clock-hole of a tower, constraints are formulated in the class *ClockHole*. This means that the information needed to construct a *ClockTowerModel* is hereby placed in three separate models, the class diagram for *Clock* (placed in the package *ClockPk*), the class diagram for *TowerModel* (placed in the package *TowerModelPk*) and in the package *ClockTowerModel*. In the package model for the clock tower only related classes from the *ClockPk* and *TowerModelPk* packages need to be shown, since the information of the three models are merged when implementing the information in a PCS.

9) Allow the display of classes, which belong to multiple wholes or inherits from more than one class without having to show the same class more than once in the diagram: When dealing with classes that are parts in more than one whole-part relationship or inherit from more than one class, these classes have to be drawn several times in a PVM-model, which can be avoided by using class diagrams. This is exemplified in figure 11, where the class *Wheels* is a part of two wholes and consequently drawn in two places in the PVM, while it is shown only once in the class diagram. The problem of PVMs also exists if a class inherits from two or more classes, which also implies that the same class must be shown several times in a model. Furthermore, in cases where a generalisation class is not part of a whole-part relationship, this cannot be expressed within the main model, and the model must be divided into two parts, as seen in figure 11, where the class *Axle* is placed outside the main model.

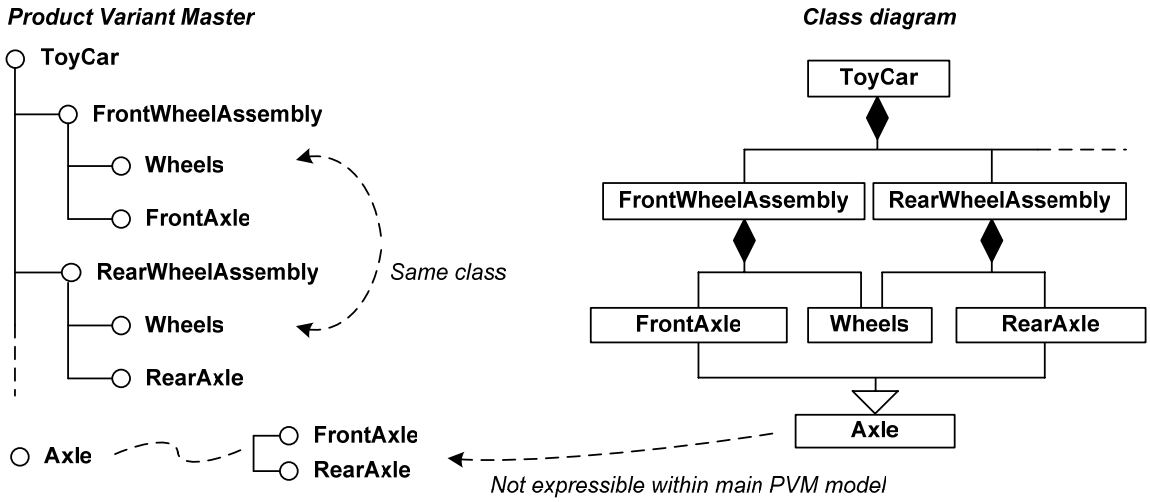


Figure 11: Dealing with multiple inheritance and whole classes in PVMs and class diagrams

The problem of PVMs having to show the same classes several times in a model can be avoided by using the PSCD technique, as shown in figure 12.

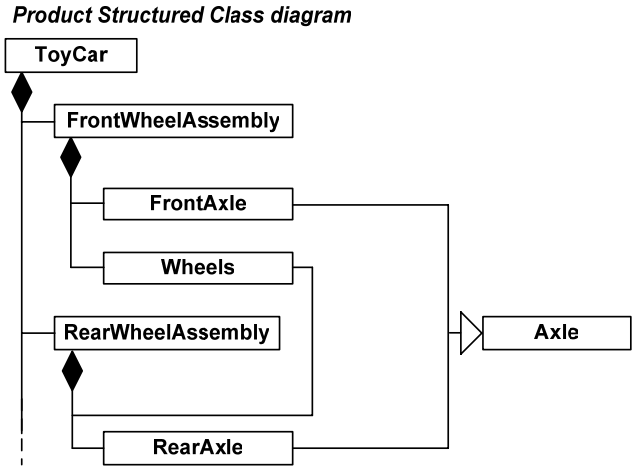


Figure 12: Dealing with multiple inheritance and whole classes in PSCDs

To sum up, the utility analysis of PSCDs compared to class diagrams and PVMs have showed that PSCDs to a great extend holds the nine mentioned advantages of class diagrams compared to PVMs.

5. Usability analysis

5.1 Choice of experiment

As mentioned Haug and Hvam (2006) provide two overall explanations of the seemingly unanimous agreement, which is found in literature and industry, that PVMs are more user-friendly than class diagrams: 1) The fixed placement of classes, i.e. the placement of different relationship types in separate columns in a PVM and PVMs, and the hierarchical placement of aggregation classes, which implies a reading pattern similar to the one of text, making PVMs and PSCDs easier to review than class diagrams; and 2) More easily understandable symbols, primarily by the use of a hierarchical list for symbolising aggregation as opposed to the diamond symbol in class diagrams. Therefore, if PSCDs are to have user-friendliness similar to PVMs, this diagram must hold the mentioned characteristics, and this is exactly was PSCDs do.

To illustrate the assertion that it is easier to review PVMs and PSCDs than normally drawn class diagrams, the review pattern of such diagrams are illustrated in appendix 1. It is seen that the review pattern for the aggregation classes of a PSCD model is from left-to-right and a move down when there is no more elements on the current horizontal line. On the other hand, when reviewing normally drawn class diagrams, the review pattern includes both right, left and down movements, while moving up to resume the review. For large and complex models represented in class diagrams this obviously makes the review task more difficult.

However, while PSCDs utilise the user-friendly fixed positioning of elements from PVMs, there are some differences in appearance between the two diagrams. The difference in appearance between a model drawn by using PVMs and PSCDs is that: on a PSCD model class names and class contents are placed within a box whereas on a PVM this information is placed next to a circle; the two diagrams apply different generalisation symbols; the display of the aggregation symbol (black diamond) on PSCDs; and minor differences in the way cardinality and constraints are represented. These layout-related differences could have an impact on how easy a PSCD model is to understand and overview compared to a PVM model, but under the assumption that the two mentioned overall explanations of Haug and Hvam (2006) hold, this difference seems to be less significant and mainly related to individual preferences. Hereby not to say that this is not an issue that should be further investigated, but as a starting point it seems more interesting to focus on how easy PSCD models are to construct compared to PVM and class diagram models. To investigate this, an experiment was carried out.

5.2 Knowledge engineer experiment

In order to investigate the usability of PSCDs compared to PVMs and class diagrams from a novice knowledge engineer perspective an experiment was carried out in October 2006. The experiment had the purpose of investigating how easy models are to create when using PSCDs, PVMs and class diagrams, including how fast you can build a model, how many errors are made in the process, and how easy a created model is to read. Due to the relatively small number available of participants (engineering students with the required prerequisites), the experiment was to a certain extend perceived as a pilot experiment.

5.2.1 Participants

Eighteen engineering students participated in the experiment. The students were all attending an end-level course (last two years of the five-year long M.Sc.) about product configuration at the Technical University of Denmark. As a part of the course, prior to the experiment, the students had all had lectures about how to PVMs and class diagrams and had created models by using these types of diagrams in connection with exercises. However, the students had never been introduced to PSCDs.

5.2.2 Procedure

Firstly, ten minutes was spent on briefly resuming PVMs and class diagrams and on giving an introduction to PSCDs. The eighteen students were then divided into nine groups to allow discussions while elaborating the models. The groups were given a textual description of the variance of the mountain bikes of an imaginary bicycle manufacturer. The model that was to be built consists of 54 different classes in an aggregation hierarchy, at least nine specialisation classes (classes with individual information), 27-40 attributes (depending on whether types are shown as attributes or specialisation classes), and 16 constraints. Three groups were told to use PVMs, three groups to use class diagrams and three groups to use PSCDs to create the model. The constraint expressions were not to be rewritten on the models, but references to the constraints were to be shown next to relevant attributes, i.e. several instances of a reference for constraints that involves several attributes. The groups were given 40 minutes to get as far as they could. The models were drawn with pencils on large sheets of paper. During the experiment the students were not given any help other than for understanding the assignment text.

5.2.3 Results and discussion

In figure 13 some of the results of the experiment are shown.

	PVM				PSCD				Class diagram			
	G1	G2	G3	*Mean*	G4	G5	G6	*Mean*	G7	G8	G9	*Mean*
Classes drawn	60	34	70	*54,7*	36	69	62	*55,7*	37	76	53	*55,3*
Attributes drawn	41	14	25	*26,7*	12	44	30	*28,7*	7	42	19	*22,7*
Constraint references drawn	14	9	11	*11,3*	12	10	12	*11,3*	0	0	7	*3*
Constraint reference errors	3	2	0	*1,7*	2	1	3	*2*	0	0	2	*0,7*
Other errors	2	0	1	*1*	0	2	0	*0,7*	4	4	2	*3,3*
Model width [cm]	30	30	30	*30*	30	30	30	*30*	42	152	61	*85*
Model Length [cm]	128	42	107	*92,3*	42	98	139	*93*	30	30	30	*30*

Figure 13: Results of knowledge engineer experiment

Looking at the mean values, the only two things that really stand out are the differences in the low number of *constraints references drawn* and the higher number of *other errors* in the class diagram models compared to PVMs and PSCDs. *Other errors* include: not creating specialisation classes with individual attributes, stating attribute variance incorrect, and placing classes incorrect. The problems in placing the constraint references in class diagrams compared to PVMs and PSCDs are in line with the assumption that this would be more difficult in class diagrams. However, the small sample size, combined with the large variations in the groups within one notation, makes these results somewhat unsuited for making final conclusions. It seems that the groups varied much in modelling skills, although they had received the same amount of education. This implies that a larger sample would be needed in order to be able to make solid conclusions about the levels of difficulty of the different types of diagrams.

On the other hand the differences in the layout of the produced models provide an interesting observation. Here the placement rules in PSCDs seem to have contributed to much more well-organised representations than the class diagram models. This is shown in appendix 2, where extracts from the two most completed models from the PSCD and class diagram groups are placed.

6. Conclusions

Three approaches with different prescriptions for the use of structural diagrams for the development of PCSs were identified in literature. The first approach included PVMs as the only structural diagram, the second only included class diagrams, and the third approach prescribed the use of PVMs followed by class diagrams. While class diagrams provide a richer, more formalised and more flexible modelling language than PVMs, it seems that PVMs holds two major advantages over class diagrams, namely that these seem to be easier to learn and better suited for pointwise reviews. This means that if only PVMs or class diagrams are applied in a PCS project this could give some limitations, as it might turn out that class diagrams require thorough training of domain experts, or that PVMs are not adequately rich or formalised to describe a design. A solution could, therefore, be to do as the CPM-procedure proposes, namely use PVMs for domain analysis and class diagrams for the creation of design models. Although this approach has significant advantages as compared to the application of just one of the diagrams for both tasks, there is also a downside, namely that information has to be transferred from PVM models to class diagram models. This process can be time-consuming and may result in errors, which may be difficult to spot.

Trying to capture the advantages of both PVMs and class diagrams in a new diagram type, Haug and Hvam (2006) suggested that some of the special characteristics of PVM diagrams were applied in class diagrams, naming this technique PSCDs. The use of PSCDs instead of PVMs followed by class diagrams means that a transfer of information between analysis and design models can be avoided. However, Haug and Hvam (2006) did not investigate if PSCDs hold the advantages of both PVMs and class diagrams, for which reason this investigation was carried out in this paper.

Based on (Haug and Hvam, 2006) nine advantages of using class diagrams compared to PVMs were presented. It was shown that the implications of imposing the restrictions for the placement of the model elements of PVMs on class diagrams did not result in significant limitations compared to

the use of class diagrams without these restrictions. The nine mentioned advantages of class diagrams over PVMs can, therefore, also be perceived as advantages of PSCDs over PVMs.

The paper presented an experiment with the purpose of investigating certain usability characteristics of PSCDs as compared to class diagrams and PVMs. Eighteen engineering students divided into nine groups participated in the experiment, where, based on a textual product description, three groups were asked to create models by using PVMs, three groups to use class diagrams and three groups to use PSCDs. However, due to the small sample size and great variations within the samples the data from the experiment do not provide an adequate basis for making final conclusions. Anyhow, strong indications that the placement-rules of PSCDs make these easier to draw were obtained. But in order to provide strong evidence of that the usability of PSCDs are at the level of PVMs and have higher usability than class diagrams further experiments are required.

Despite the lack of solid evidence from experiments, it does seem, from a common sense point of view, that the usability of PSCDs is quite similar to the usability of PVMs and therefore PSCDs are as well-suited for modelling sessions in configuration projects as PVMs. Since PSCDs also hold the advantages that class diagrams have over PVMs, it seems that the use of PSCDs in many cases would be a better approach than the use of PVMs or class diagrams for the creation of both analysis and design models or the use of PVMs followed by class diagrams.

References

Arlow, J. and Neustadt, I. (2005). *UML 2 and the Unified Process* (2nd edition). Upper Saddle River, NJ: Addison Wesley.

Beck, K. and Cunningham, W. (1989). A laboratory for teaching object-oriented thinking. *SIGPLAN Notices*, Vol. 24, No. 10: 1-6.

Cilia, M., Haupt, M., Mezini, M. and Buchmann, A. (2003). The convergence of AOP and active databases: Towards reactive middleware. In *Proceedings of 2nd International Conference on Generative Programming and Component Engineering* (GPCE), Erfurt, Germany, 22-25 Sept, *LNCS 2830*. New York: Springer-Verlag: 169-188.

Edwards, K. and Ladeby, K. (2005). Framework for Assessing Configuration Readiness. In *Proceedings of the World Congress on Mass Customization and Personalization* (MCPC 2005), Hong Kong, 18-21 Sept. Hong Kong: HKUST.

Evermann, J. and Wand, Y. (2005). Toward formalizing domain modeling semantics in language syntax. *IEEE Transactions on software engineering*, Vol. 31, No. 1: 21-37.

Felfernig, A., Friedrich, G.E. and Jannach, D. (2000). UML as domain specific language for the construction of knowledge based configurations systems. *International Journal on Software Engineering and Knowledge Engineering*, Vol. 10, No. 4: 449-470.

Felfernig, A., Friedrich, G. and Jannach, D. (2001). Conceptual modeling for configuration of mass-customizable products. *Artificial Intelligence in Engineering*, Vol. 15, No. 2: 165-176.

Forza, C. and Salvador, F. (2002). Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems. *International Journal of Production Economics*, Vol. 76, No. 1: 87-98.

Forza, C., Trentin, A. and Salvador, F. (2005). Product Information Management for Mass Customization: the Case of Kitting. In *Proceedings of the 3rd Interdisciplinary World Congress on Mass Customization and Personalization* (MCPC2005). Hong Kong, 18-21 Sept. Hong Kong: HKUST.

Fowler, M. (1997). *UML Distilled - Applying the standard object modeling language*. Reading: Addison-Wesley.

Fowler, M. (2005). *UML Distilled* (3rd edition). Boston, MA: Addison-Wesley.

Grudin, J. (1992). Utility and Usability: Research Issues and Development Contexts. *Interacting with computers*, Vol. 4, No. 2: 209-217.

Hansen, B., Riis, J. and Hvam, L. (2003). Specification process reengineering: concepts and experiences from Danish industry. In *Proceedings of the 10th ISPE international Conference on Concurrent Engineering: Research and Applications*, Madeira, Portugal, July 26-30. A.A. Balkema Publishers.

Harlou, U. (2006). *Developing product families based on architectures - Contribution to a theory of product families*. PhD thesis. Lyngby, Denmark: Department of Mechanical Engineering, Technical University of Denmark.

Haug, A. and Hvam, L. (2006). A comparative study of two graphical notations for the development of product configuration systems. Accepted (Aug. 2006) for publication in *International Journal of Industrial Engineering*.

Hvam, L. (1994). *Application of product modelling - seen from a work preparation viewpoint* (Trans). PhD thesis. Lyngby, Denmark: Department of Industrial Management and Engineering, Technical University of Denmark.

Hvam, L. (2004). A Multi-perspective approach for the design of Product Configuration Systems - An evaluation of industry applications. In *Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems*, Lyngby, Denmark, 28- 29 June. Lyngby, Denmark: Technical University of Denmark, pp. 13-25.

Hvam, L., Mortensen, N.H. and Riis, J. (2007). *Produktkonfigurering*, [Product Configuration]. Copenhagen, Denmark: Nyt Teknisk Forlag.

Hvam, L., Riis, J. and Hansen, B.L. (2003). CRC cards for product modelling. *Computers in Industry*, Vol. 50, No. 1: 57-70.

Kolp, M., Giorgini, P. and Mylopoulos, J. (2002). Information Systems Development through Social Structures. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, Ischia, Italy, 15-19 July. ACM Press, pp. 183-190.

Mortensen, N.H., Yu, B., Skovgaard, H. and Harlou, U. (2000). Conceptual modeling of product families in configuration projects. *Workshop at the 14th European Conference on Artificial Intelligence*, Berlin, Germany, Aug. 20-25. Electronic proceedings available at: www.cs.hut.fi/pdmg/ECAI2000WS/Proceedings.pdf, pp. 68-73.

Männistö, T., Soininen, T. and Sulonen, R. (2001). Modelling Configurable Products and Software Product Families. In *Proceedings of 7th International Joint Conference on Artificial Intelligence* (IJCAI '01). Seattle, Washington, USA, Aug. 4-10. Available at: http://www.soberit.tkk.fi/tmm/

Nielsen, J. (1992). The usability engineering life cycle. *Computer*, Vol. 25, No. 3: 12-22.

OMG (2005). *Unified Modeling Language: Superstructure* (Version 2.0: Formal/05-07-04). www.uml.org.

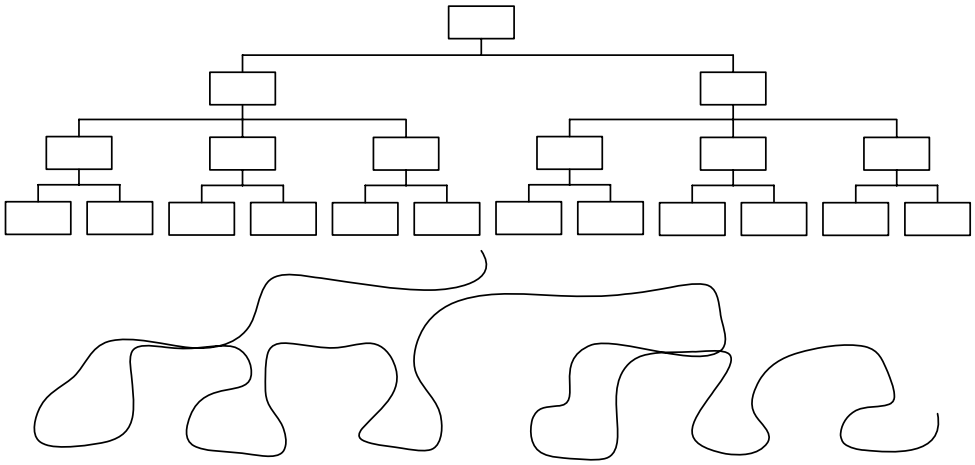
Priestley, M. (2003). *Practical Object-Oriented Design with UML* (2nd edition). New York: McGraw-Hill.

Riis, J. (2003). *Fremgangsmåde for opbygning, implementering og vedligeholdelse af produktmodeller - med fokus på konfigureringssystemer*, PhD thesis. Lyngby, Denmark: Department of Manufacturing Engineering and Management, Technical University of Denmark.

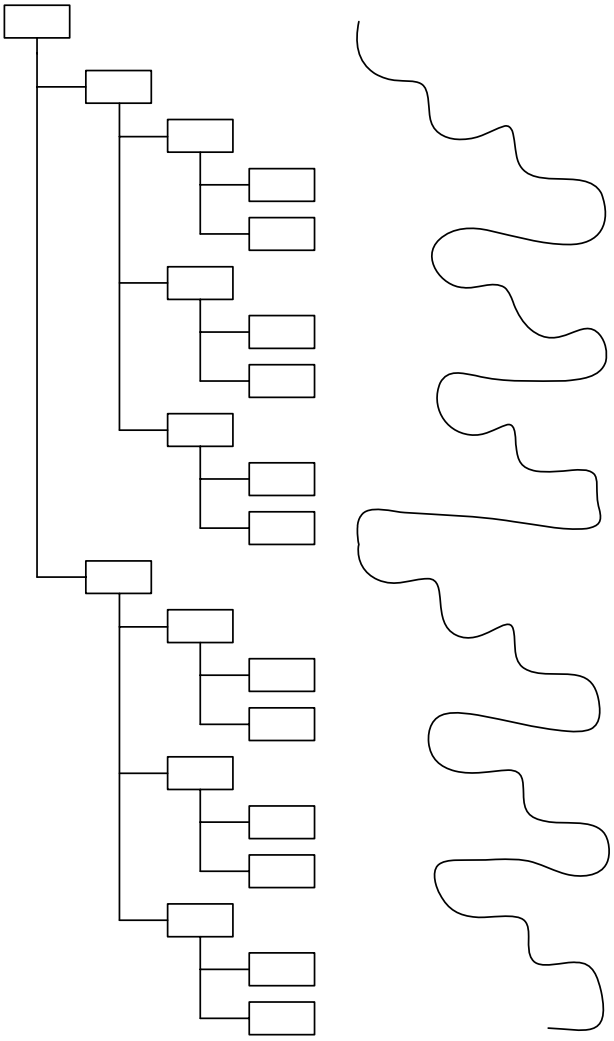
Rozen, S. and Shasha, D. (1989). Using a Relational System on Wall Street: The Good, the Bad, the Ugly, and the Ideal. *Communications of the ACM*, Vol. 32, No. 8: 988-994.

Sabin, D. and Weigel, R. (1998). Product Configuration Frameworks - A survey. *IEEE Intelligent Systems & Their Applications*, Vol. 13, No. 4: 42-49.

Appendix 1: Review patterns - Class Diagrams vs. PSCDs

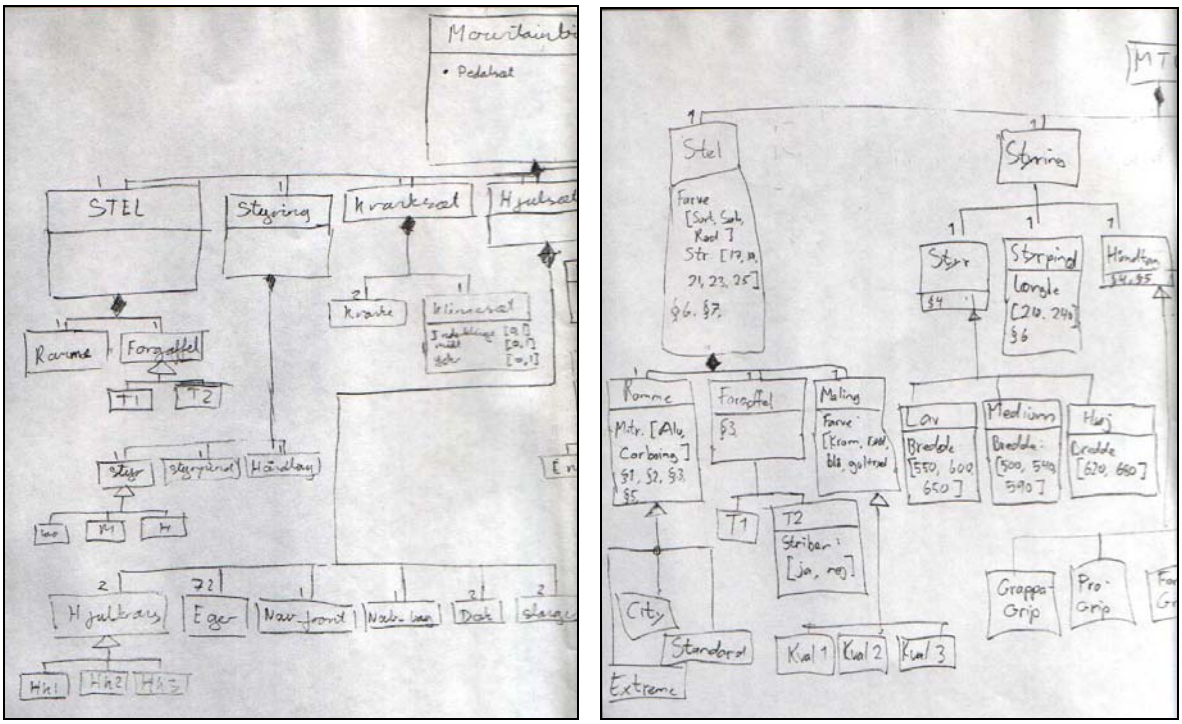


Normally drawn class diagrams and review pattern

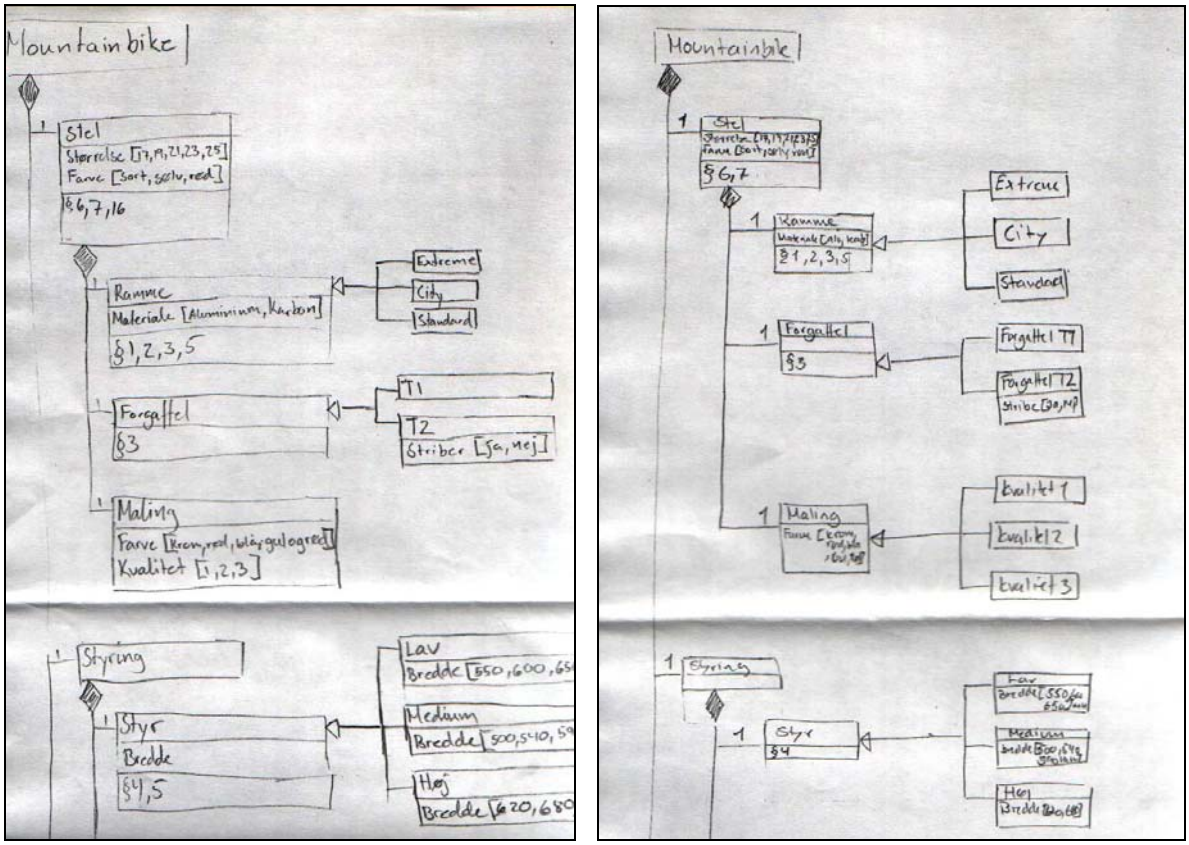


PSCD aggregation hierarchy and review pattern

Appendix 2: Experiment - Class Diagrams vs. PSCDs



Extract from groups using Class Diagrams



Extract from groups using PSCDs

Paper B4

Merging models with different perspectives on product configuration knowledge

Haug, A. and Hvam, L. (2006): "Merging models with different perspectives on product configuration knowledge", in Research in Interactive Design, Volume 2, Springer-Verlag, France (Proceedings of Virtual Concept, Cancun, Mexico, Nov. 26 - Dec. 1, 2006).

Merging models with different perspectives on product configuration knowledge

Anders Haug and Lars Hvam

Abstract: Product configuration systems (PCS) can be defined as product-oriented expert systems that allow users to specify products by selecting components and properties under restriction of valid combinations, i.e. an interactive design process. The application of configuration technology has for several years produced such benefits as reduction of lead times, resources and errors. Developing PCSs implies that domain knowledge is represented. In itself, the representation of domain knowledge can be a big challenge, but this task can be further complicated when there is a need to handle models with different perspectives on domain knowledge. In the paper, it is argued that models with different perspectives on product configuration knowledge often include both individual and overlapping content, and that maintaining such models can be time-consuming and lead to errors. To solve this problem, a concept that allows such models to be maintained within a common model is proposed.

Keywords: Product configuration, product modelling, knowledge representation, knowledge engineering, documentation of configuration systems.

1- Introduction

Product configuration systems are a successful application of techniques from the field of artificial intelligence. A PCS can be defined as a product-oriented expert system (or knowledge based system), which allows users to specify products by selecting components and properties under restriction of valid combinations, i.e. an interactive design process. Benefits from configuration projects have been described in the literature [FS1, FT1, R1, H1, H2, HN1]. This literature, among others, mentions such benefits as: lower lead times (sales to delivery), improved quality of product specifications, preservation of knowledge, use of fewer resources for specifying a product, optimized products, less routine work, improved certainty of delivery, and less time needed for training new employees.

Before implementing domain knowledge in a PCS, models are most often created by using different knowledge representation techniques. The representation of domain knowledge is often one of the greatest tasks in a configuration project [SW1, HR1, EL1]. The task of creating or maintaining representations of domain knowledge can be further complicated when applying models with different perspectives on a domain. In many cases, such models include both individual and shared content, which produces a need for ensuring consistency across the models when updating documentation. Since this kind of work can be time-consuming and result in errors, a modelling technique that addresses this problem would be beneficial. Therefore, this paper proposes a concept that allows models that have different perspectives on product knowledge, while sharing some of their contents, to be maintained without having to document the same information twice, thereby avoiding the problems of ensuring consistency across models.

In this paper, two kinds of model classifications are in focus: (1) analysis and design models, and (2) models representing different configuration stages. In brief, analysis and design models, respectively, describe a real life domain of interest and how this should be implemented in a PCS. Models that represent different configuration stages refer to situations where a PCS should support different specification processes, for instance, during sales and later during detailed design. Besides these two classifications, also a third can be identified, namely, models that describe different views on a product from a design perspective, e.g. function view, property view, module view, and part view. In some cases, this kind of model view corresponds to *configuration stage models*, while in other cases such views on product knowledge are merely applied to be able to understand or create clear descriptions of the product knowledge. Views on a product from a design perspective are not explicitly dealt with in this paper, but the proposed modelling technique would also be applicable in contexts where these kinds of models have overlapping content.

The rest of the paper is structured as follows: In section 2, two types of scenarios where there is a need to maintain separate models with overlapping content are analysed. Next, in section 3, two commonly applied knowledge representation techniques in configuration projects are described. In section 4, a modelling technique that allows a merger of models with overlapping content is proposed. In section 5, experience with using the modelling technique in practices is described. The paper ends with a conclusion in section 6.

2- Models with different perspectives and shared information

2.1 - Analysis and design models

Within software development, two characteristic kinds of models exist, namely analysis models and design models. Analysis models (also called conceptual models and domain models) are created early in a software project, before initiation of design and coding. The purpose of analysis models is to describe and understand a real-life domain independently of the choice of software; therefore, the focus is more on investigating than creating a solution [P1, L1]. A design model can be perceived as an elaboration of the analysis model; therefore, a design model normally contains much of the same information but includes more details and makes references to properties of the proposed software system [P1, L1].

The concept of analysis and design models used in general software development is also applied in some projects where the task is to create a PCS. In this context, analysis models are used to describe product-related knowledge, not only with the intent of describing the knowledge that is later to be implemented into the knowledge base of a PCS, but also to provide a basis for discussions of: what is to be included in the project, which parts of the product assortment are suited for configuration, which parts of the product should be standardized etc. Describing the real-life domain in a way that enables these discussions can in itself be challenging, and since the inclusion of implementation issues at this stage results in additional complexity, the distinction between analysis and design models can be very useful.

The product descriptions that have been produced during the analysis phase are not necessarily suited for direct implementation, as a model may have to be redesigned in order to be supported by the chosen software. Also, in some cases, analysis models can be simplified without losing information, which could ease implementation and maintenance of product knowledge in a PCS. The use of design models therefore implies a more fluent transition from model to implementation into the PCS knowledge base. Since, in some configuration projects, only parts of what is described in the analysis model are to be implemented at a specific stage, design models can be applied in order to define this selection, while the rest of the analysis model is saved for possible later implementation.

When a design model has been implemented and the PCS has been taken into use, the project moves into the maintenance phase. In this phase, the design model can be used as external documentation and should in this case be updated when changes occur. Alternatively, the modelling environment of the PCS has features that can be used for documentation of the implemented design. Unless the analysis model is also updated when changes occur, a discrepancy between the implemented knowledge and the analysis model emerges. This can be problematic when changes of the PCS knowledge base require the involvement of domains experts who are only familiar with the analysis language. A possibility for avoiding such situations is to maintain both analysis and design models. However, it can be difficult and time consuming to keep these consistent, especially if they are created using different modelling languages. The ideal situation in many cases would therefore be to have a single model in a modelling tool that is capable of providing both an analysis view and a design view. According to [AN1], no UML (Unified Modelling Language) modelling tool currently on the market does an entirely satisfactory job of providing both analysis and design views of the same underlying model. They therefore describe four strategies for managing analysis and design models, all of which, however, have significant disadvantages. These are shown in figure 1.

Strategy	Consequences
1. Refine analysis model into a design model	The analysis model is lost
2. Refine analysis model into design model and use modelling tool to recover analysis model	The recovered analysis model might not be satisfactory
3. Freeze analysis model at some point, and refine a copy of the analysis model into a design model	The two models are out of step
4. Maintain both analysis and design model	Maintenance burden

Figure 1: Maintenance strategy (adapted from [AN1])

2.2 - Configuration stage models

Some PCSs support different specification processes in a company. The phrase *configuration stage models* refers to models that are applied in different specification processes and often by users from different domains.

In [HN1], three perspectives on a product family are described: customer view, engineering view, and part view. In brief, the customer view covers the aspects that are of interest to the customer; the engineering view describes the product functions and the appurtenant variance; while the part view describes physical components (either bought or produced) that the product family consists of. In [HN1], examples of how these views could be applied are provided; however, in these examples the views are modelled separately, although some model elements appear across models.

Other ways of classifying perspectives on a product family exist, for instance, the *theory of dispositions* [O1]. The theory of dispositions focuses on the relation between a design and the life-phase activities of a product. Dispositions refer to the parts of decisions that are taken within one functional area and affect the type, content, efficiency or progress of activities of other functional areas. [O1] mentions twelve systems that a product can meet during its life: planning, fabrication, assembly, testing, transport, sales, installation, operation, service, scrapping, recycling and deposition. Although the focus in this context is on mass-produced products, these views could also be relevant in a configuration project, but in a different order.

The configurations that need to be created by a PCS at different stages of a project can be even closer related than in the described divisions, for instance, in a scenario where a PCS should support sales, preliminary design, and detailed design. During the sales phase, maybe only some basic components or modules are chosen in order to calculate an estimated price. Later, if an offer is accepted, preliminary design of the product can be initiated. This phase would result in a more detailed selection and specification of components. Next, a detailed design phase can be initiated, which results in further specification of the product. Creating a PCS that supports these configuration stages may require that a sales model, a preliminary design model, and a detailed design model are created. Besides these models having individual contents, they might also share some content between them. For instance, the knowledge needed to generate a price estimate could be included in the sales model, while the final price calculations are made outside the PCS in the detailed design phase, just as the preliminary design model and detailed design model would include details that are not considered at the sales configuration stage.

In figure 2, an illustration of how three models in principle can be interconnected is shown. Here, the nodes symbolize model elements and the lines relations between these. Such interconnectivity means that changes of one model can impact one or both of the other models, which can make the maintenance of such models complicated.

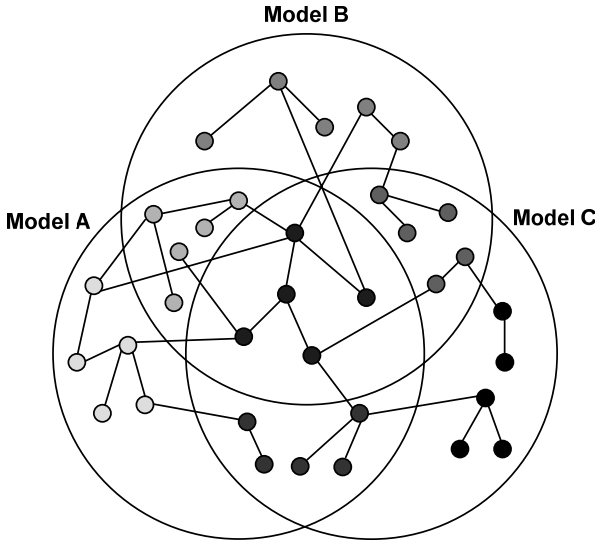


Figure 2: Three interrelated models with shared elements

3- Knowledge representation techniques in configuration projects

As a basis for describing the proposed modelling technique that addresses the problem of documenting models with overlapping content, this section briefly describes product variant masters (PVM) and class diagrams. PVMs and class diagrams are part of a procedure for the development of product configuration systems that has been applied in several Danish configuration projects [H2, HN1]. The procedure prescribes the use of PVMs in the analysis phase, based on the experience that these are more easily understood by domain experts than class diagrams. On the other hand, class diagrams provide a richer and more formalized language, which in many cases makes these better suited for the creation of design models. Extended CRC-cards (class, responsibility, and collaboration) are prescribed for keeping the detailed descriptions of the classes of PVMs and class diagrams in order to enhance the clarity of these structural representations. The original CRC-cards are described in [BC1], and descriptions of the extended CRC-cards used in the mentioned procedure for the development of PCSs can be found e.g. in [HR2, HN1].

3.1 - Product variant masters

The PVM technique provides a formal way of representing a product assortment. A PVM consists of two generic sections, *part-of structure* and *kind-of structure*. The part-of structure defines the parts a given product family can comprise, while the kind-of structure describes the variation of a part, i.e. different types with common characteristics. In object-oriented terms these two structures roughly correspond respectively to aggregation (or composition) and generalization. The latest definition of the PVM technique is found in [HN1] and is shown in figure 3.

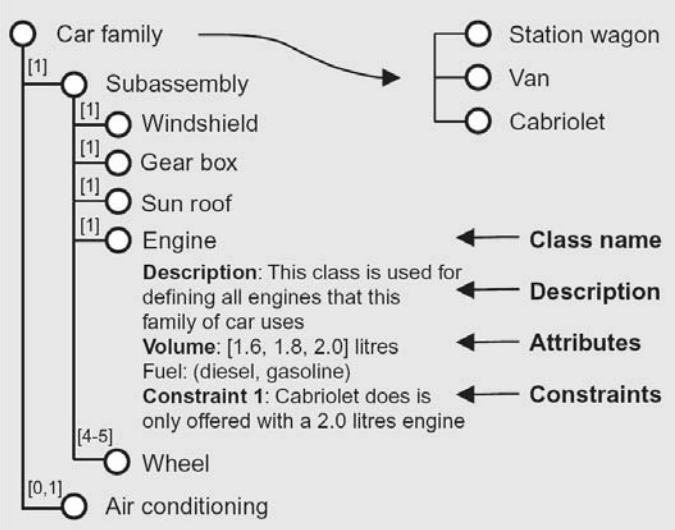


Figure 3: The most recent definition of the PVM formalism ([HN1] from [H3])

PVMs are normally drawn by using software such as MS Visio or Excel. The models are intended to be printed out on big sheets of paper and are refined during repeated sessions where knowledge engineers and domain experts discuss and define the products in focus [H2, HN1].

3.2 - Class diagrams

Compared to PVMs, class diagrams are much more widespread in use and have a broader range of applications. Class diagrams are one of thirteen diagram types in UML 2.0 [O2] and the most used of these [F1]. Class diagrams depict the static structure of a system by describing the objects and the relationships between these. The notation for the class element and the most common relationship types are shown on figure 4, where a *navigability arrow* can be used to show the direction of association, aggregation and composition relationships.

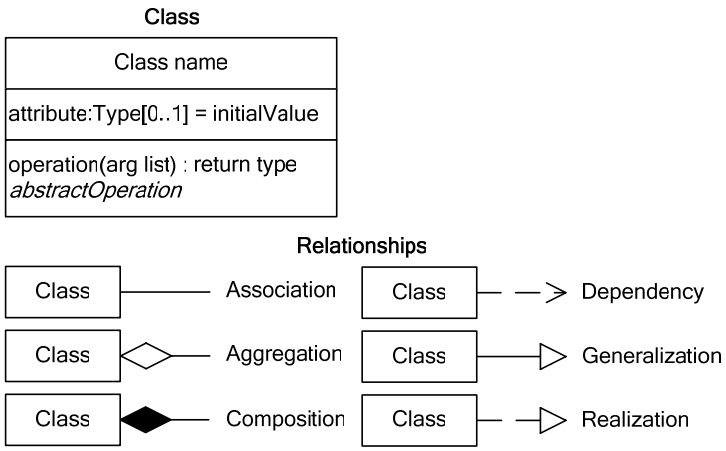


Figure 4: Commonly applied elements of class diagrams

To describe constraints, UML provides a formal language called OCL (Object Constraint Language), but any formalism for the description of constraints is allowed as long as it is placed within braces ({}) [F1]. Another important concept in UML is stereotypes. Stereotypes allow the introduction of new model elements based on existing elements. This is done by appending the stereotype name in guillemets (<<...>>). Stereotypes can be collected in profiles together with tagged values and constraints to be used for customizing UML for a specific domain. For further descriptions of class diagrams, see e.g. [F1, O2].

4- Merging models of different views

To deal with the problems arising from having models with different perspectives on product knowledge but with some shared content, a modelling technique is proposed in this section. The core principle of the modelling technique is to merge models with overlapping content (i.e. classes, attributes, methods, constraints, and relationships) into a common model, and then classify the contents of the common model according to which model view it belongs to. If for instance a *model A* and a *model B* should be merged into a common model, the contents of the common model should be classified according to whether it belongs to *model view A*, *model view B*, or both views. Looking at the shared content together with the content that only belongs to *model view A* therefore gives a view corresponding to *model A*, while looking at the shared content together with the content that only belongs to *model view B* gives a view corresponding to *model B*. Sharing model content between the different model views means that changes of shared content in one view automatically impacts the other view; therefore, in such a case there is no need for ensuring consistency across models. In order to make this principle operational, solutions on two levels are required: a modelling technique, and a software solution to support this technique.

4.1 - A modelling technique for merging models

When creating a common model by merging different models together, UML stereotypes can be applied for classifying the contents of this common model according to which model view it belongs to. If it is assumed that the two overlapping models, *model A* and *model B*, are merged into a common model, the content of this common model should then be stereotyped according to whether it belongs to: only *model view A*, only *model view* B, or both views. For this, only two stereotypes are needed, as not stereotyped model content would imply that it is shared, i.e. the stereotypes <<*a*>> and <<*b*>>.

However, this classification is not adequate, since classes of one model view could be represented as attributes in another model view. If a basis is taken in a situation with a *model A* and a *model B*, where the class, *ClassA*, of *model A* is represented as the attribute, *attributeB*, in *model B*, the obvious solution would be to stereotype *ClassA* with <<*a*>> and *attributeB* with <<*b*>>. This would mean that *ClassA* and *attributeB* are only shown in the model view to which they belong. However, if

ClassA is renamed, deleted or obtains new attributes, similar changes may have to be made in *model B*. But, since there is no indication of whether the <<*a*>> stereotyped *classA* only exists in *model A* or if it is represented in another form in *model B*, *model B* would have to be examined. To avoid this, the stereotypes <<*cta*>> (class to attribute) and <<*atc*>> (attribute to class) are introduced to indicate whether a class is represented as an attribute in another view or the other way around (if there are more than two views, the naming of these stereotypes must be extended to indicate which of the two views the transformation occurs between). The use of the <<*cta*>> and <<*atc*>> stereotypes means that all classes, attributes, methods, constraints, and relationships can now be changed while knowing whether a similar change might be needed in other model views.

In UML, the multiplicity of a class is stated on its relationships to other classes, while the multiplicity of an attribute is shown in square brackets ([..]) placed after the attribute name. This means that if a class in one model is transformed into an attribute in another model, the multiplicity stated on the class' relation in the first model should be stated behind the attribute name in the other model.

Another transformation indicator is needed for situations where classes in one model view are merged together in another model view, and the merged classes carry attributes, methods or constraints. Just keeping the original name of attributes, methods and constraints may result in classes that hold attributes, methods or constraints of the same name. For instance, if two classes *Nut* and *Bolt* in a *model A* both include the attribute *colour*, and these two classes in a *model B* are merged into a *NutAndBolt* class, this means that the *NutAndBolt* class includes two attributes of the name *colour*, but it is impossible to know what these *colour* attributes refer to. So to ensure a unique and more meaningful naming of attributes, methods, and constraints in such situations, these can be named by combining the name of their original class followed by underscore and their attribute name. In the example this would mean that the class *NutAndBolt* includes the attributes *nut_Colour* and *bolt_Colour*.

In one model view, a class can have specialization classes, while this generalization class in another view should be represented as an attribute. In this case, the specialization classes can be transformed into possible values of the attribute created from the generalization class. For instance, if *ClassA* has the specialization classes, *Type1* and *Type2*, this could after the transformation be represented as the attribute, *classA {Type1, Type2}*. If the specialization classes hold the same attributes, methods or constraints, these could be shown by using the original class names followed by underscore and the attribute name, e.g. *type1-2_Colour*. On the other hand, if the attributes, methods or constraints of the specialization classes are different, a more complex solution is required. If it is assumed that a *ClassA* has the two specialization classes, *Type1* and *Type2*, where *Type1* holds the attribute *colour*, while *Type2* holds the attribute *surfaceType*, then first *ClassA* is transformed into an attribute and the specialization classes into possible values of *ClassA*, i.e. *classA {Type1, Type2}*. As either *Type1* or *Type2* must be chosen, only one of the attributes *colour* and *surfaceType* is relevant. To handle this, these attributes can be shown with the multiplicity *[0,1]*, while adding a constraint that tells that the multiplicity is 0, if the type to which an attribute belongs is not chosen.

As defined, the attributes in a model view, which have been moved from other classes in another model view, have a name that starts with the name of their original class, followed by underscore. As the use of underscore indicates that the attribute originates from another class in another model view, the <<*atc*>> stereotype could be left out, while the <<*cta*>> stereotype should still be used in the other model view. Therefore, the <<*atc*>> stereotype is in principle only needed when a class in a *model A* is merged with its specialization classes and turned into an attribute in a *model B*.

4.2 - Example of the use of the modelling technique

To illustrate the described modelling technique, an example is shown in figure 5. The figure displays a part of an analysis model and the corresponding part of the design model. In the example, PVMs are applied for analysis and class diagrams for design, but obviously PVMs or class diagrams could be used for both models. Stereotypes are not included in the PVM definitions but are added to the

notation in this context. In the class diagrams of the design model, it is chosen to place constraints within the class symbol as opposed to placing them in a note symbol, which is the most common way of doing this. This is done to limit the number of model elements in order to enhance the clarity of the representation. After the attributes, their possible values are written in guillemets ({..}).

In the example in figure 5, the classes *Screwbolt*, *Plate* and *Hook* from the analysis model are incorporated into the class *PlateHookAssembly* in the design model, while the analysis class *Grinding* is left out of the design model. In the analysis model, the classes *Screwbolt*, *Plate* and *Hook* are stereotyped <<*cta*>>, as they in the design model only exist as part of the name of their attributes. Their class names could have been shown in the design model (e.g. *Screwbolt [1]*), but as this does not add any information, it is omitted. The attribute *Screwbolt_Diameter* is the only attribute in the design model that has its multiplicity shown (*[2]*), as this multiplicity differs from the default value of 1.

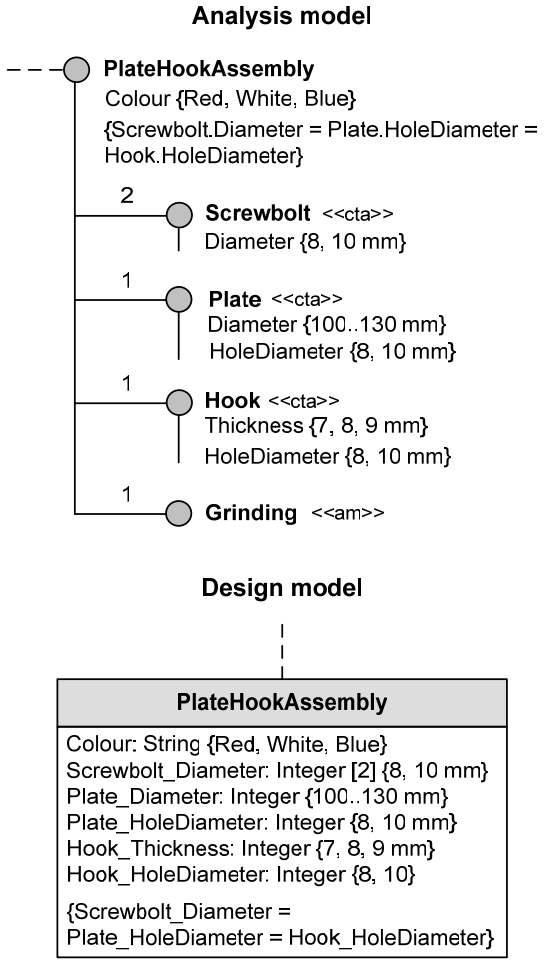


Figure 5: Analysis and design model view

In the example in figure 5, only part-of/aggregation relationships are included. However, product models often also include kind-of/generalization relationships. To illustrate how to handle kind-of/generalization relationships, figure 6 shows parts of the analysis view and design view of a toy car model. In the example, the specialization classes of the class *Chassis* are incorporated into their generalization class by being represented as attribute values of the attribute *chassis*. Since the specialization classes *Type1* and *Type2* include attributes with different value options, both attributes are shown with a multiplicity of *[0,1]*. Furthermore, a constraint is added to handle these multiplicities, depending on which type of chassis is chosen. The classes *Body* and *SideSticker* in the analysis model are represented as attributes of the class *BodyAssembly* in the design model; therefore, their specialization classes are converted into attribute values. As the multiplicity of the class

SideSticker in the analysis model differs from 1, this multiplicity (*[0,2]*) is shown after the attribute *sideSticker* in the design model. The class *Screw* and the attribute *Length* of the class *Chassis* in the analysis model are omitted in the design model, as they have the analysis model stereotype. On the other hand, a class *PriceCalc* is introduced in the design model.

It should be noted that the attributes of the classes *Type1*, *Type2* and *Body* of the analysis model do not carry the <<*cta*>> stereotype, although they change class in the design model. This is because when a class is stereotyped <<*cta*>>, this implies that its attributes, methods, and constraints are represented in another class in the design model; therefore, stereotyping these as well would be superfluous. The same applies to specialization classes of <<*cta*>> stereotyped classes, where the stereotyping of their generalization class implies that the specialization classes are represented as attribute values in the design model; therefore a stereotyping of these specialization classes is superfluous.

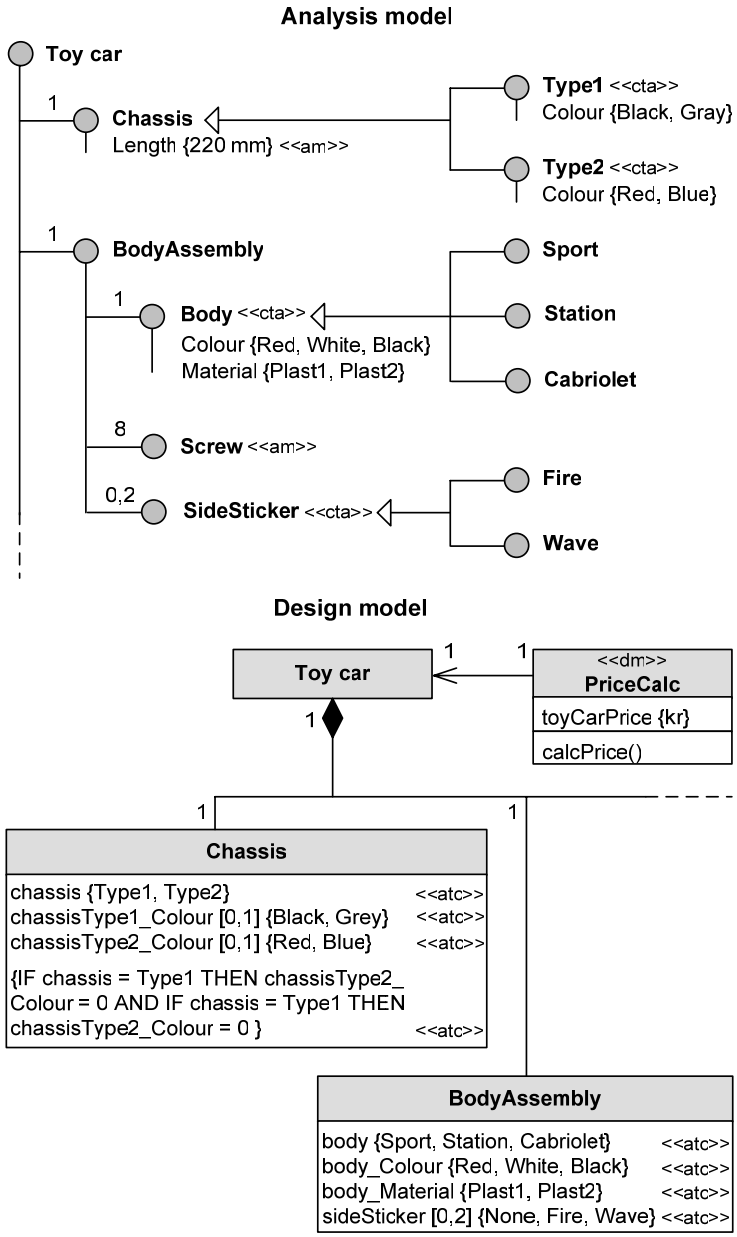


Figure 6: Analysis and design model view of toy car model

4.3 - Software support of the modelling technique

Having defined a modelling technique that supports the idea of merging models with overlapping content, the question that remains is how this technique can be used in practice, i.e. which software supports this. Obviously, two alternatives exist, namely the use of existing software or the creation of special software.

MS Visio is commonly applied software for the elaboration of graphical models within PCS development. Visio includes a layer functionality that allows the application of the proposed modelling technique. The idea is that the contents of a total model are placed in different layers that represent the required views. Different views can therefore be obtained by turning layers on and off. However, using the layer solution in Visio requires that the same notation is used in both the analysis and design models. Taking the analysis and design models from figure 6, these could be represented in a merged model, as shown in figure 7, where it is presumed that class diagrams are used for both the analysis and design models. Turning off the analysis model layer would result in a model view similar to the design model in figure 6, while turning off the design model layer would result in the analysis model. To indicate whether a class is turned into an attribute from the analysis model to the design model or the other way around, <<*cta*>> and <<*atc*>> stereotyped classes, attributes and constraints also include a model stereotype in the total model, i.e. <<*am*>> or <<*dm*>>. To enhance the clarity of the analysis and design model views, the stereotype symbols can be placed in a layer of their own, which allows these to be turned off, while still being able to see the desired model view.

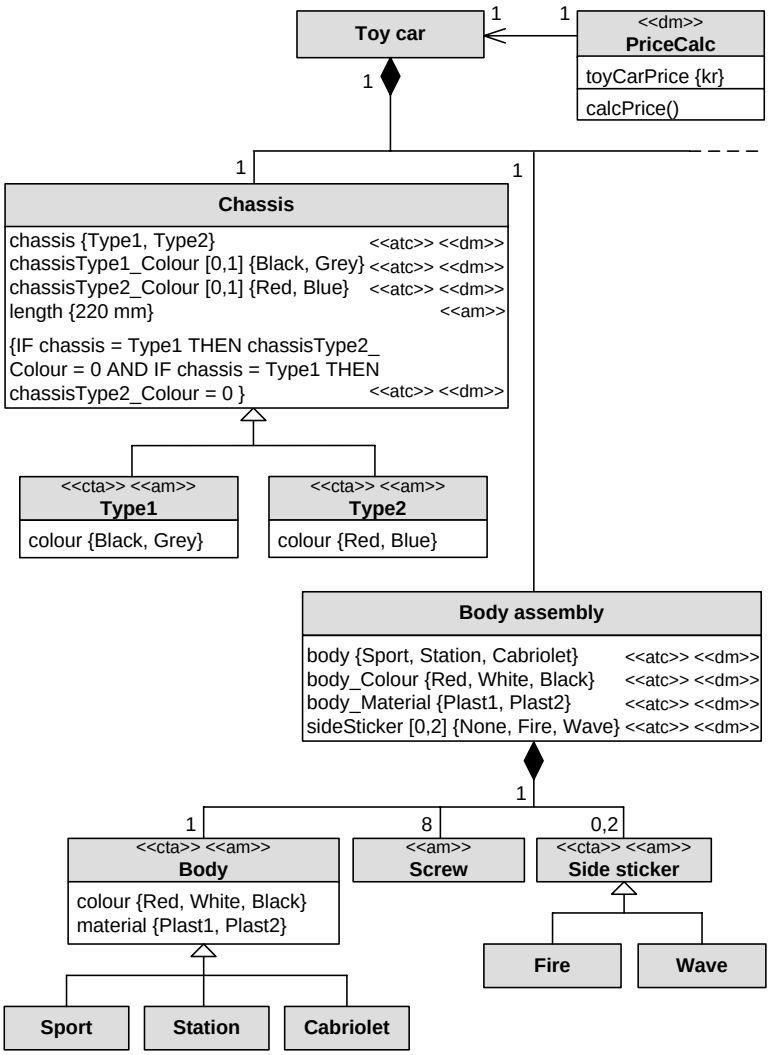


Figure 7: The total model of toy car

The Visio solution, however, has a significant limitation in that the placement of elements in the total model defines the placement in both views. This means that classes, attributes, constraints, and methods leave blank spaces in the views where these are turned off. In a case where the use of individual model elements is limited, this might not be a big problem, but extensive use of individual elements could cause a confusing representation. A better solution would therefore be to use more *intelligent* software for this kind of modelling.

The creation of a documentation system that supports the development and maintenance of PCSs is a topic that has been subjected to some research. In [HM1], requirements for a documentation system that supports PVMs, class diagrams and CRC-cards are provided. They further describe the creation of a prototype, which however only supports the CRC-card techniques together with a hierarchical list. Later in [HP1], additional requirements for a documentation system are defined, and a high-level description of the architecture of such a system is presented. Recently, detailed descriptions of how the notations of PVMs, class diagrams and CRC-cards could be mapped to each other and represented in the user interface of a documentation system have emerged in [HH1]. This is supplemented by an extended CRC-card definition in [HH2].

At the Centre for Product Modelling at the Technical University of Denmark, the creation of such a system is an ongoing project. Based on the definitions in [HH1, HH2], a prototype was created in 2006. This prototype supports the elaboration of PVMs, class diagrams and CRC-cards based on a common data model, meaning that the same model can be viewed in any of these three kinds of representations. The prototype includes placement rules so that classes in the PVM view are placed automatically. This kind of solution thus eliminates the problem of blank spaces in models when applying MS Visio. That the model views of the prototype are created based on a common data model, instead of as in Visio on a common graphical model, also opens the possibility of merging analysis models and design models, which are drawn by using PVMs and class diagrams, respectively.

5- Empirical study

Having provided an example of the use of the modelling technique applied to analysis and design models, the empirical study concerns a case with models of different configuration stages. The project in focus is an ongoing project concerning the development of a PCS to support the specification of balconies. As this is currently an ongoing project, the company wishes to remain anonymous. The focus of the project is to create a PCS that supports configuration of both sales specifications and engineering design specifications. The intended future process is that sales persons use the PCS to create a sales configuration that produces the tender material. If the tender is accepted, manual strength calculations should be carried out by engineers in order to detail or correct the defined solution principles of the tender specification. Based on these calculations, an engineering design configuration should be carried out in the PCS to produce engineering design specifications.

At the present time only an analysis model, which includes both a sales and an engineering design view, has been elaborated. The analysis model was created using extended PVM notation (e.g. by allowing modelling of multiple inheritance) and was drawn in MS Visio. The model consists of more than a thousand classes, attributes and constraints, while more than 90 percent of the contents of the sales and engineering view is identical. The main difference between the two views is that the engineering view includes more components and dimensions to choose from. Therefore, creating and maintaining this information in two separate models would imply that much manual work should have been carried out to ensure consistency across the two models. The application of the proposed modelling technique in the current case was therefore not done to test the modelling technique, but rather to avoid having to handle two big overlapping models.

The analysis model was created together with domain experts who had no modelling experience. Therefore, it was important to limit the complexity of the models. Also, since there was no real need for transforming classes to attributes between the two views, the <<*cta*>> and <<*atc*>> stereotypes

were not applied. Only two stereotypes for defining which view the classes and relationships belong to were used, <<*sales*>> and <<*engineering*>> respectively. These two views were modelled in a common model, where content that only belongs to either the sales or the engineering view is placed in different layers. In this way, by turning off one of the layers, it is possible to print out both views from the same model. Figure 8 outlines the principle that was applied in the described case.

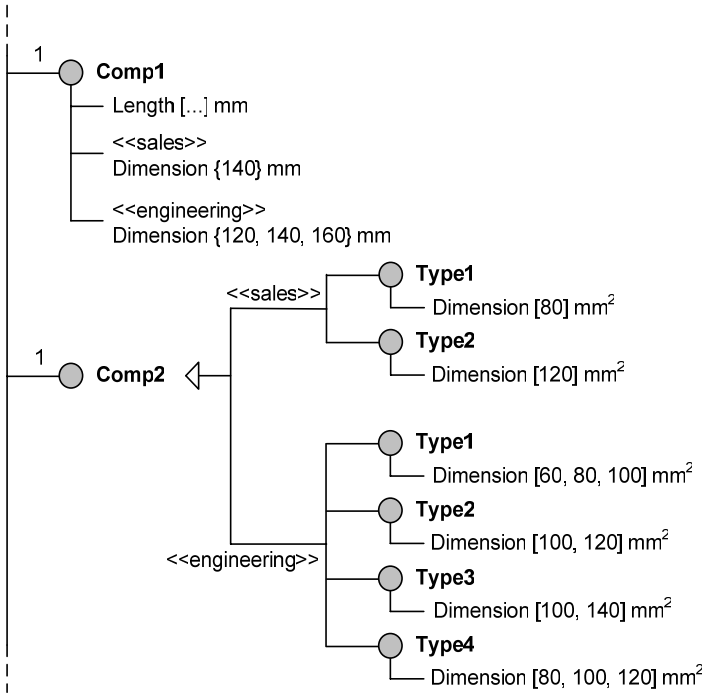


Figure 8: Principle illustration of the modelling technique as applied in practice

The experience from the application of the modelling technique was positive in the sense that the number of models to be handled was in principle reduced by half. The blank spaces created by the turned off classes, attributes, and relationships were not commented by any of the involved domain experts during the modelling sessions.

6- Conclusions

In product configuration projects, there is sometimes a need for having models with different perspectives on product knowledge. In this paper, two types of scenarios were analysed, one where both analysis and design models are needed, and one where configuration is to be performed at different stages of a project. In both types of scenarios, the created models often share some content, which means that the creation and maintenance of these models requires that consistency across models is ensured. This can be time-consuming and a possible source of errors. To address this problem, the paper proposes a modelling technique that allows a merger of different models with overlapping content into a common model. The solution includes two levels, firstly a modelling technique, and second a definition of how this technique could be supported by software.

The core of the proposed modelling principle is to merge models with different views on product knowledge, while stereotyping classes, attributes, methods, constraints, and relationships according to which model view they belong to. This implies that the contents of the total model include both shared content, and content that only belongs to a particular model view. The modelling technique also includes definitions of how to handle classes and their attributes, methods, and constraints when such classes are represented as attributes in other model views. Furthermore, a definition of how to handle specialization classes, which in another model view should be included in their generalization class, is included.

Two software solutions were discussed. The first is a MS Visio solution where the layers of Visio are used for separating content belonging to different model views. These layers can then be turned on and off, depending on the desired model view. However, the two major disadvantages of the Visio solution are that turned off classes, attributes, methods, constraints, and relationships leave blank spaces in a model, and that the models must be created by using the same representation technique. The second solution is to include the modelling technique in a documentation system to support the development and maintenance of PCSs, which is currently an ongoing project at the Technical University of Denmark. A prototype of such a system has been developed. The prototype generates its different model views based on a common data model instead of, as in Visio, a common graphic model. This solution therefore supports creating models by using different representation techniques; and because the prototype includes rules for how classes and relationships are placed, the problem of blank spaces in a model view is eliminated.

The application of the proposed modelling principle in an ongoing configuration project was presented. The experiences from the project show that in some cases the use of MS Visio provides adequate software support, and reduces the need for having separate models with overlapping content. In the studied project, however, only minor parts of the total model belonged to individual model views, and only a part of the proposed modelling technique was applied. Therefore, experience from further applications of the proposed modelling technique is required to determine whether adaptations or extensions are needed.

In any case, the inclusion of the modelling technique into a PCS documentation system could be the necessary means to allow companies to enjoy the full benefits from the application of the proposed modelling technique.

References

[**AN1]** Arlow J. and Neustadt I. UML 2 and the Unified Process (2nd edition). Addison Wesley, Upper Saddle River, NJ, 2005.

[BC1] Beck K. and Cunningham W.A. A laboratory for teaching object-oriented thinking. In SIGPLAN Notices, 24(10): 1-6, 1989.

[EL1] Edwards K. and Ladeby K. Framework for Assessing Configuration Readiness. Proceedings of the MCPC 2005, Hong Kong, Sept 18-21 2005.

[F1] Fowler M. UML Distilled (3rd edition). Addison-Wesley, Boston, MA, 2005.

[FS1] Forza C. and Salvador F. Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems. In International Journal of Production Economics 76: 87-98, 2002

[FT1] Forza C., Trentin A., Salvador, F. Product Information Management for Mass Customization: the Case of Kitting. Proceedings of the MCPC 2005, Hong Kong, Sept 18-21 2005.

[H1] Hansen B.L. Development of Industrial Variant Specification Systems, PhD thesis, Department of Manufacturing Engineering and Management, Technical University of Denmark, 2003.

[H2] Hvam L. A Multi-perspective approach for the design of Product Configuration Systems - An evaluation of industry applications. Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems, Lyngby, Denmark, June 28-29, 2004.

[H3] Harlou U. Developing product families based on architectures – Contribution to a theory of product families. Dissertation, Department of Mechanical Engineering, Technical University of Denmark, 2005.

[HH1] Haug A. and Hvam L. The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems. In Customer Interaction and Customer Integration, Series on Business Informatics and Application Systems, 2: 165-181, 2006.

[HH2] Haug A. and Hvam L. CRC-cards for the development and maintenance of product configuration systems. In Customer Interaction and Customer Integration, Series on Business Informatics and Application Systems, 2: 149-164, 2006.

[HM1] Hvam L. and Malis M. A Knowledge Based Documentation Tool for Configuration Projects. Proceedings of World Congress on Mass Customization and Personalization, Hong Kong, Oct. 1-2, 2001.

[HN1] Hvam L., Mortensen N.H. and Riis J. Produktkonfigurering (Preliminary edition for education purposes, first edition is to appear later in 2006). [Product configuration]. Nyt Teknisk Forlag, Copenhagen, Denmark, 2006.

[HP1] Hvam L., Pape S., Jensen K.L. and Riis J. Development and maintenance of product configuration systems - Requirements for a documentation tool. International Journal of Industrial Engineering, 12(1): 79-88, 2005.

[HR1] Hansen B., Riis J. and Hvam L. Specification process reengineering: concepts and experiences from Danish industry. Proceedings of the 10th ISPE International Conference on Concurrent Engineering: Research and Applications, Madeira, Portugal, July 26-30, 2003.

[HR2] Hvam L. and Riis J. CRC cards for product modelling. In Computers In Industry, 50(1): 57-70, 2003.

[L1] Larman C. Applying UML and Patterns (2nd edition). Prentice Hall, Upper Saddle River, NJ, 2002.

[O1] Olesen J. Concurrent Development in Manufacturing - based on dispositional mechanisms. PhD thesis, Institute for Engineering Design, Technical University of Denmark, 1992.

[O2] OMG (The Object Management Group). Unified Modeling Language: Superstructure (Version 2.0: Formal/05-07-04). www.uml.org, 2005.

[P1] Priestley M. Practical Object-Oriented Design with UML (2nd edition). McGraw-Hill, New York, 2003.

[R1] Riis J. Fremgangsmåde for opbygning, implementering og vedligeholdelse af produktmodeller - med fokus på konfigureringssystemer. [Procedure for building, implementing and maintaining product models - with focus on configuration systems]. PhD thesis, Department of Manufacturing Engineering and Management, Technical University of Denmark, 2003.

[SW1] Sabin D. and Weigel R. Product Configuration Frameworks - A survey. IEEE Intelligent Systems & Their Applications, 13(4): 42-49, 1998.

Paper C1

The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems

Haug, A. and Hvam, L. (2007): "The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems", International Journal of Mass Customisation, Issue 1/2, Vol. 2.

The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems

Anders Haug and Lars Hvam

Abstract: This paper deals with the subject of creating a documentation system to support the development and maintenance of Product Configuration Systems (PCSs).

A procedure for building PCSs from the Centre for Product Modelling (CPM) at the Technical University of Denmark has, for more than ten years, been applied in numerous projects. The CPM procedure includes three main modelling techniques for development and maintenance of PCSs. Since no single software supports all three techniques, there is no automatic integration between the different kinds of models, which means that some information has to be modelled repeatedly. This presents a demand for a coherent documentation system that supports the modelling techniques of the CPM procedure. In this paper, an important step towards fulfilling the ambition of creating such a system is taken by presenting a redefinition of the included modelling techniques and a definition of their mutual mappings.

Keywords: product modelling; product configuration; knowledge representation; documentation of product models.

1 Introduction

To meet customer demands for customised products while keeping the costs low, Product Configuration Systems (PCSs) can be applied. A PCS can be defined as a product-oriented expert system, which allows users to specify products by selecting components and properties under the restriction of valid combinations. The application of configuration technology can produce benefits such as shorter lead times, reduction of resources needed to produce specifications and fewer errors in specifications (Hvam, 2004).

The development of a PCS implies a relocation of knowledge from domain experts to a software system. This means that domain knowledge has to be represented, which is one of the greatest challenges of a configuration project (Sabin and Weigel, 1998; Hansen *et al.*, 2003). The development of PCSs can also imply the defining of new domain knowledge, since when a PCS project is started, the included product families often do not have a well-considered permanent architecture or a basis for creating a robust generic product model (Pulkkinen, 2000). Great involvement of domain experts is therefore often necessary when developing PCSs.

To support the development of PCSs, several approaches have been proposed. In this paper, the focus is on a procedure from the Centre for Product Modelling (CPM) at the Technical University of Denmark, which includes three major techniques for modelling configuration knowledge. In the development phase, models are normally drawn by using programs such as MS Visio, Excel and Word or Computer Aided Software Engineering (CASE) tools such as IBM's Rational Rose. The use of different kinds of software means that there is no automatic integration between the models. Therefore some information has to be modelled repeatedly, which is time consuming and may lead to errors.

Over time, the products of a company change, which means that the knowledge base of the PCS has to be updated for the PCS to support the company processes. As a PCS can easily have a knowledge base that consists of thousands of interrelated components, constraints and methods, and the modelling environment of a PCS often does not provide an adequately comprehensible overview of the implemented knowledge, getting an overview that allows updates to take place often requires external documentation.

The described aspects present a demand for a coherent documentation system that supports the modelling techniques of the CPM procedure, which at the present time does not exist. Research on how to create this documentation system has been carried out in recent years (Hvam and Malis, 2001; Hvam *et al.*, 2005a). That research offers a list of requirements, including the modelling techniques to be supported, but does not explain in detail how these modelling techniques should be presented in the user interface, and, what is even more critical, how the different modelling techniques should be mapped to each other. Existing research, therefore, does not provide an adequate basis for creating a documentation system that supports the required modelling techniques. To fulfil this need this paper presents such definitions. However, this paper does not deal with functional requirements in detail, which to some extent has been provided by earlier research (Hvam *et al.*, 2005a).

In Section 2, the CPM procedure is outlined with a focus on its techniques for the modelling of product knowledge. Next, in Section 3, previous research concerning the documentation of PCSs is resumed. In Section 4, a definition of a documentation system to support the development and maintenance of PCSs is presented. The paper ends with a conclusion in Section 5.

2 The CPM procedure and applied modelling techniques

2.1 The CPM procedure

The CPM procedure was introduced by Hvam (1994) and has since undergone further development at CPM. The CPM procedure has been applied in several projects, which in many cases has led to major improvements in lead times, the number of errors, the use of resources, *etc.* (Hvam *et al.*, 2002; Hvam, 2004). The CPM procedure embraces many aspects of a product configuration project and provides guidelines for its seven phases: (1) (business) process analysis, (2) product analysis, (3) object-oriented analysis, (4) object-oriented design, (5) programming, (6) implementation and (7) maintenance. In the product analysis phase, the prescribed modelling technique is the so-called Product Variant Master

(PVM). In the succeeding object-oriented phases, the PCS is defined by using Unified Modelling Language (UML) class diagrams as the primary notation.

The object-oriented analysis phase includes a transfer of PVMs to class diagrams, which is not intended to be done in a one-to-one manner but with optimisation of the models in mind. The basic argument for using both modelling techniques instead of just one of them is that the PVM notation offers a relatively easily comprehensible syntax, which, as experience has shown, is easily accepted by domain experts; on the other hand, class diagrams are richer and more flexible, though still more formalised, wherefore this notation is better suited for the handling of complex designs. Also, the widespread use of UML within software development is an important factor. To allow more detailed specifications of the classes represented in diagrams, the CPM procedure proposes the use of special CRC (Class, Responsibilities and Collaborators) cards.

2.2 Product Variant Masters (PVMs)

As mentioned, the use of PVMs is prescribed in the product analysis phase of the CPM procedure. The newest version of the CPM procedure (Hvam *et al.*, 2005b) includes an updated definition of the PVM notation, originating from Harlou (2005). This is shown in Figure 1.

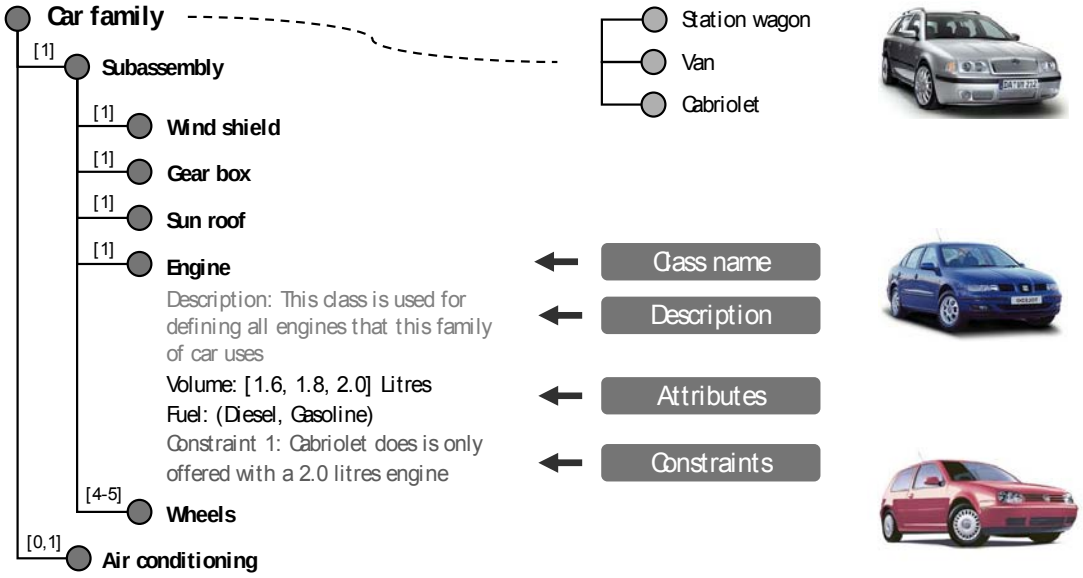


Figure 1: Definition of the PVM-notation (Hvam et al., 2005b)

A PVM consists of two sections, on the left side the *generic part-of structure* and on the right side the *generic kind-of structure*. The part-of section describes the elements of which a product can consist, while the kind-of section describes the possible variance of an element.

A PVM model is intended to be printed on a large sheet of paper and be discussed in repeated sessions of model-managers (often industrial engineers) and domain experts, leading to continuous refinements of the model.

Experience from the use of the PVM technique (based, however, on earlier definitions of the notation) has shown that a PVM can be a useful tool for discussing where to start modelling the PCS, which product variants to include, the stability of products, *etc.*(Hvam *et al.*, 2002; Hvam, 2004).

2.3 Class diagrams

In the object-oriented analysis phase of the CPM procedure, the problem domain and the field of application in which the IT system will be used are analysed. In the succeeding design phase, the focus changes towards implementation issues, such as design of user interfaces, adaptation of analysis model and data management. A main purpose of the object-oriented phases is to formalise the model created in

the preceding product analysis phase in a way that allows it to be implemented in a configuration system. To do so, UML diagrams, primarily class diagrams, have been chosen as the modelling technique.

The CPM procedure (Hvam *et al.*, 2005b) prescribes the use of a subset of the class diagram notation, which is shown in Figure 2. For the full notation see OMG (2005).

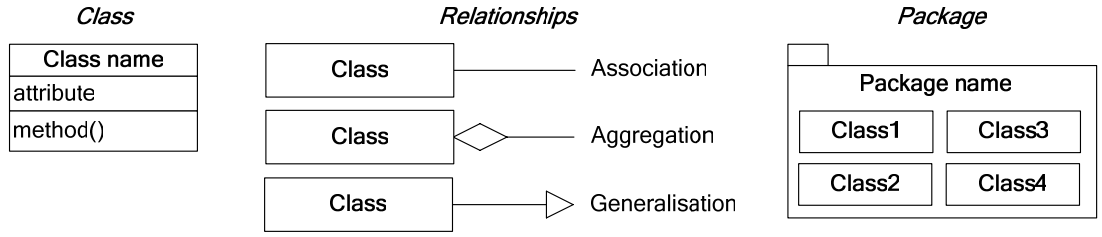


Figure 2: The selection of class diagram model elements of the CPM-procedure

According to Hvam *et al.* (2005b), the kind-of and part-of relations of a PVM can be translated into *generalisation* and *aggregation* relationships, respectively, in a class diagram. Aggregation is in this context perceived in a broad sense, not having differentiated between *aggregation* and *composition* (symbolised by a black diamond). In this paper, the composition relationship is applied, as this is the most accurate according to the definitions of the two relationship types (OMG, 2005; Fowler, 2005). Figure 3 gives a simple example of how part-of and kind-of structures can be transferred to a class diagram.

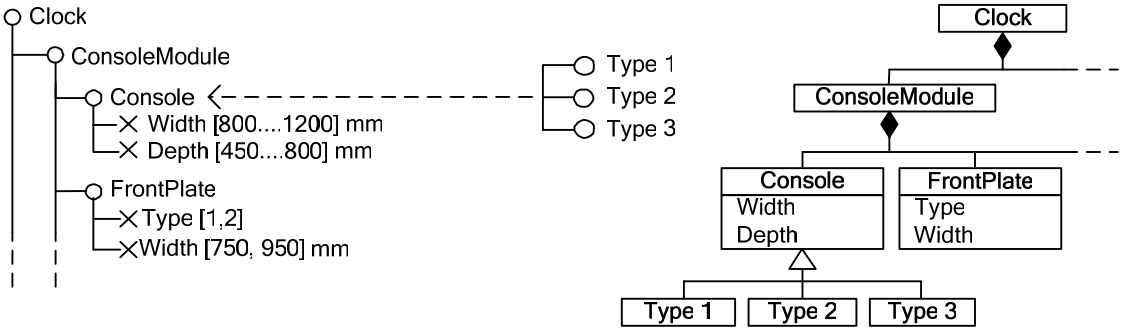


Figure 3: Transfer from a PVM to a class diagram (based on Hvam et al., 2005b)

Not included in Hvam *et al.* (2005b) but found in Riis (2003) is the UML concept of *stereotypes*, which is also found in other PCS development approaches that include class diagrams (Felfernig *et al.*, 2000). Stereotypes allow the introduction of new model elements based on existing elements (Arlow and Neustadt, 2005), which enables the use of platform- or domain-specific terminology or notation. Some stereotypes are predefined in the UML; others may be user defined. Stereotypes can be collected in a profile, which is a concept that was added in the UML 1.4 specification for extending a reference metamodel to target a specific platform or domain (OMG, 2005). Riis (2003) defines a collection of stereotypes to be applied in class diagrams in a PCS development context. This, among others, includes product family, part, function and property.

2.4 CRC cards

CRC cards were invented by Cunningham in the late 1980s (Fowler, 2005) and presented in a paper by Beck and Cunningham (1989). These CRC cards consist of the class name together with two columns for *responsibilities* and *collaborators*. In brief, responsibilities are summarisations of the things an object should do, while collaborators are the other classes with which a class must work together (Fowler, 2005). CRC cards can be seen as an alternative to structural diagrams in the early phases of a software development project, as they are a good technique for exploring object interactions by moving cards around. CRC cards are not part of UML but can be a valuable technology for learning object orientation.

CRC cards can also be applied beyond the early phases of a project for different purposes (Bennet *et al.*, 2002).

Hvam and Riis (1999) present a revised kind of CRC card to be used in product configuration projects. This CRC card definition was later updated in Hvam *et al.* (2005b), which version is seen in Figure 4. On the CRC cards, the fields *Superparts* and *Subparts* refer to aggregation relationships, while the fields *Subclasses* and *Superclasses* refer to generalisation relationships.

Class name:	Date:	Author/version:
Responsibilities:		
Aggregation		Generalisation
Superparts:		Superclasses:
Subparts:		Subclasses:
Sketch:		
Attributes:		Collaborators:
System methods:		
Product methods:		
Internal methods: External methods:		

Figure 4: The latest CRC-card definition from CPM (Translated from Hvam, 2005b)

2.5 Dealing with adapted use of the CPM procedure

Even though the CPM procedure includes a transfer from PVMs to class diagrams, Riis (2003) and Hvam *et al.* (2005b) recognise that in some projects, the creation of PVMs with matching CRC cards would be adequate, wherefore class diagrams should not be elaborated. Riis (2003) states that either PVMs or class diagrams should be chosen as final documentation, since maintaining the same information in two different diagrams would most often not be efficient.

Hvam *et al.* (2005b) distinguish between two parts of a PCS that are relevant to documentation, namely the knowledge base and other software aspects, such as user interface design and software architecture. As class diagrams are better suited for describing software aspects than PVMs, a combination of both PVMs and class diagrams for documenting different aspects could be a sensible choice in some cases. This type of approach is described by Haug and Hvam (2005), who propose a procedure for developing 3D PCSs in which PVMs are applied for the design of the knowledge base and class diagrams for the design of other PCS aspects.

3 Documentation of PCSs

As mentioned, the CPM procedure prescribes the use of PVMs, class diagrams and CRC cards during the development and maintenance of PCSs. At present, software which supports all three techniques in an integrated fashion does not exist. The lack of such a documentation system means that different kinds of software are applied. This causes a need for manual transfers of information between PVMs, class diagrams and CRC cards, which are time-consuming activities that hold the risk of errors. This aspect of product configuration has been subject to some investigations.

During 2003 and 2004, the experience gained with product configuration within 12 Danish companies was investigated in technical, economical and organisational dimensions (Edwards *et al.*, 2005). According to these investigations, the documentation task was often the first activity to be postponed or cut away from a PCS project. Doing so might save some work in the short run, but in the long run have big consequences, as several of the companies could not maintain or further develop their PCS. Another problem experienced owing to poor documentation was problems in the daily communications between persons involved in product development and PCS development.

Hvam and Malis (2001) define requirements for a documentation system to support product configuration projects and describe the development of a prototype created in Lotus Notes. Today, this Lotus Notes application is applied by the companies GEA Niro A/S and American Power Conversion A/S (APC), though in further developed versions (Hvam, 2004). Figure 5 depicts a screenshot from the Lotus Notes application at GEA Niro A/S.

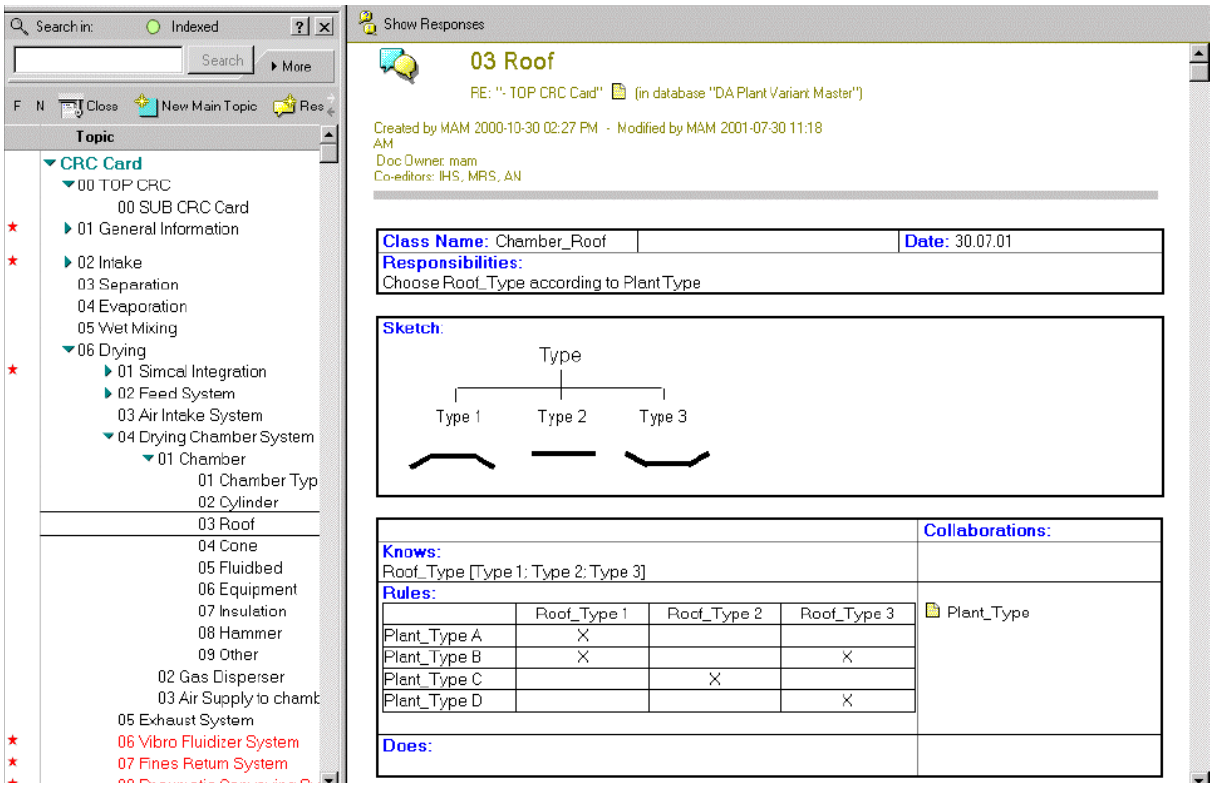


Figure 5: The documentation system of GEA Niro A/S (Hvam & Malis, 2001)

Even though the use of the Lotus Notes-based documentation system has shown that there are significant benefits of applying such a tool for the maintenance of PCSs (Hvam, 2004), much is still to be wanted. The Lotus Notes application does not support class diagrams or the full PVM notation, but offers only a hierarchical list of components with matching CRC cards. Consequently, both of the above-mentioned companies, who use the Lotus Notes application, apply other software for the creation of PVMs during the development phase.

The general impression of Hvam *et al.* (2005a) is that the CPM procedure can enhance the process of developing and maintaining PCSs, but that the manual documentation task requires too much effort as a

model grows bigger and more complex. This means that companies settle for less comprehensive and more inadequate documentation, which could change with the presence of a documentation system that supports the CPM procedure. Hvam *et al.* (2005a) therefore emphasise the need for a tool to ensure better documentation and relieve companies of the time-consuming tasks of ensuring consistent product models. To get closer to the creation of a documentation system that supports the development and maintenance of PCSs, Hvam *et al.* (2005a) present a requirement specification which is inspired by the functionality of typical CASE tools and Product Data Management (PDM) systems. The article includes a long list of requirements, but does not deal in a detailed manner with topics like user interface design, definitions of graphical notations and how to handle interrelated models. An important part of the basis for developing a documentation system to support the development and maintenance of PCSs is therefore still missing.

4 Definition of a concept for a documentation system

4.1 Method for the elaboration of a new concept

The concept for a documentation system to support the development and maintenance of PCSs, which is presented in this section, has been elaborated on, on the basis of discussions with potential users of the system and experts of software development. The companies which have provided inputs are the earlier-mentioned GEA Niro and APC, who have presented their current documentation systems and participated in discussions of ideas and requirements for a new documentation system. While elaborating on this paper, the Department of Informatics and Mathematical Modelling at the Technical University of Denmark provided feedback regarding which definitions together with existing research would provide a satisfactory basis for the development of the documentation system.

4.2 Definition of windows and navigation

Having established that the documentation system should support the use of PVMs, class diagrams and CRC cards, the question is how this functionality should be presented to the users, *i.e.*, which interfaces should be included. An obvious solution would be to create a user interface where the user creates PVMs or class diagrams in a way similar to when using Visio or Rational Rose, for example, and where a CRC card is opened by clicking on a class. This principle is very good for defining the structure of the knowledge model and should therefore be included in the documentation system. On the other hand, this kind of interface does not present the best overview of the CRC cards included in the model. For administering CRC cards the principle used in the documentation systems of GEA Niro and APC, as described in Section 3, seems to be well suited. However, this view is also not sufficient in itself owing to limitations concerning the structuring of models. The documentation system should therefore include what could be called CRC card view, PVM view and class diagram view.

In Figure 6 the defined views and the navigation between the views are depicted.

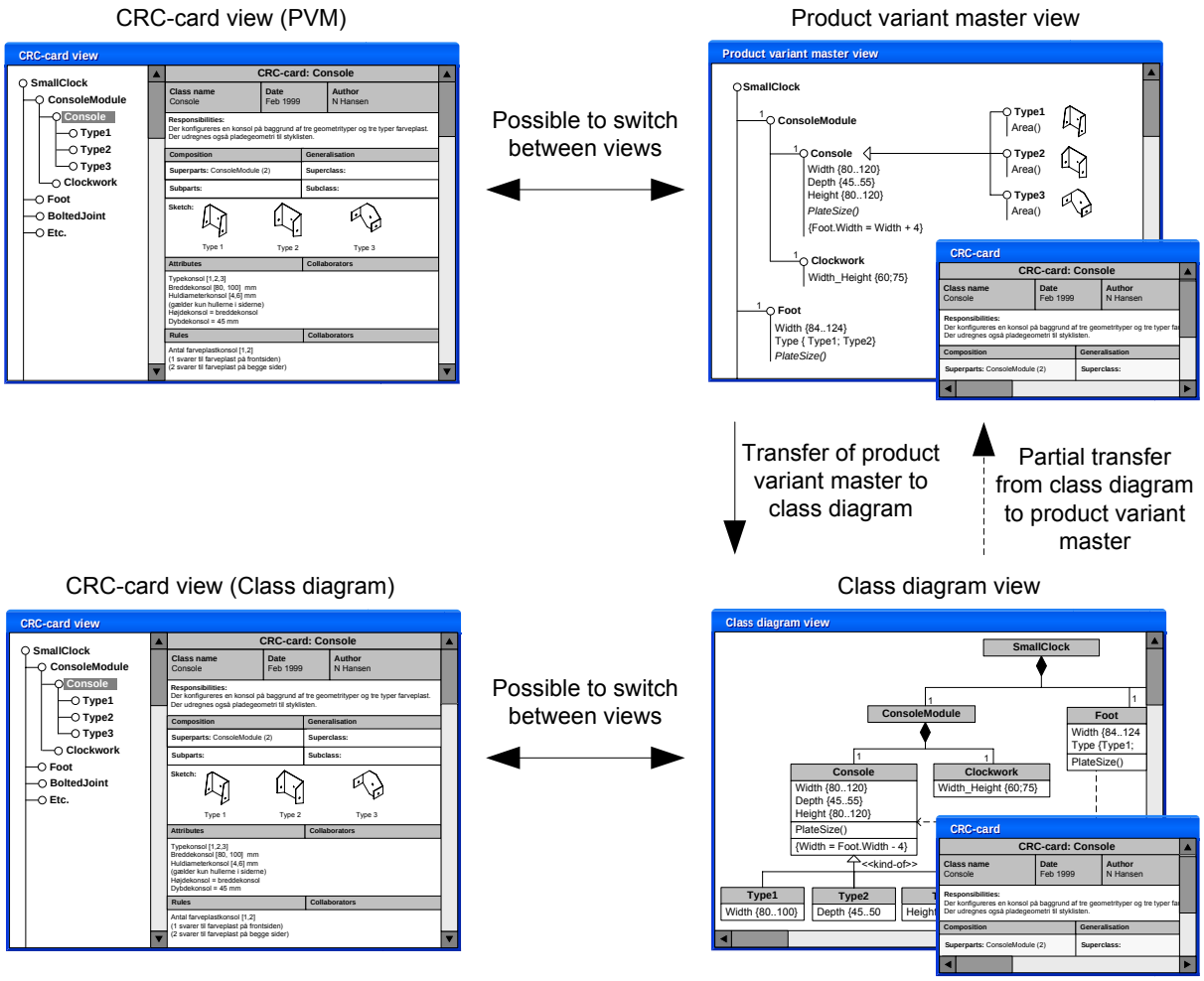


Figure 6: Windows and top level navigation in the documentation system

The documentation system should allow users to choose the model elements to be displayed in PVMs and class diagrams, for instance, to choose to hide the constraints of a class diagram.

As seen in Figure 6, a transfer from class diagram to PVM is included, though this is not defined in the CPM procedure. This functionality allows companies to go back to a more basic view, which could be beneficial if a product changes and domain experts need to be presented for a simpler representation of a model. This, however, can only be a partial transfer of model elements, since the class diagram notation is richer.

As mentioned in Section 2.5, adapted use of the CPM procedure can occur. The documentation system should therefore offer support whether PVMs or class diagrams are used as final documentation, or even both at the same time.

Besides the constraints that can be expressed graphically in diagrams, there is a need to be able to formulate further constraints. It would in most cases be advantageous to use a representation form that to some degree corresponds to the representation in the PCS.

Several representation formalisms which are employed in configuration systems exist, for instance *production rules* and *constraints* (Stumptner, 1997; Sabin and Weigel, 1998). Since class diagrams are supported by the documentation system, it is chosen to use the terms *constraints* and *methods* in the current definitions. However, it should be possible for the users of the documentation system to rename captions in CRC cards and type what they want in the constraints and methods fields.

The data model underneath the different views has to be common, as defined in earlier research (Hvam and Malis, 2001; Hvam *et al.*, 2005a). A major basis for allowing the same data to be used in different

models, created by using different notations, is that mappings between the notations are defined. These definitions are given in the following sections.

4.3 PVMs and matching CRC cards

As mentioned, different definitions of the PVM technique exist, which is why a definition of the PVM notation to be included in the documentation system is needed. In Figure 7 a PVM definition is given, which shows the minimum content to be included in the documentation system.

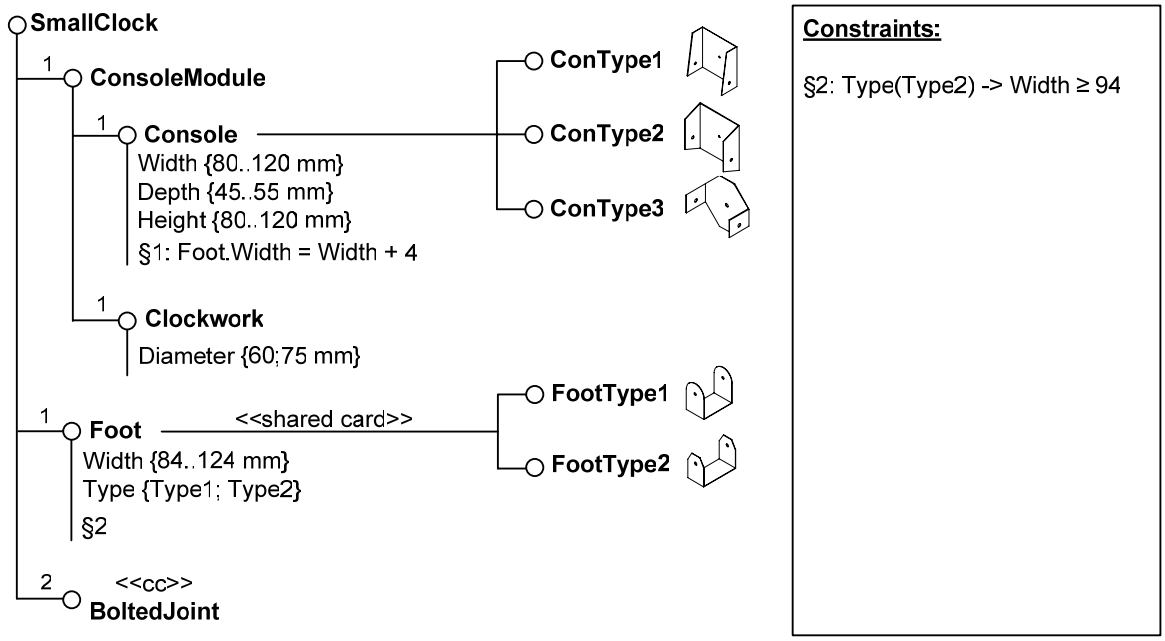


Figure 7: PVM definition

Compared to the PVM definition in Hvam *et al.* (2005b), seen in Figure 1, the following changes have been made:

- The description element within classes is found to have little use (can be found in CRC card if needed) while consuming too much space, which is why this element is excluded.
- As too many constraint expressions in a PVM can lead to a confusing representation, a constraint sheet is introduced. This allows users to use constraint identifiers (*e.g.*, §1) in the PVM and state the expressions outside the model. A constraint can therefore be expressed in the PVM (as §1) or only be shown by a constraint identifier (as §2), while stating the expression in the constraint sheet or on the matching CRC card.
- As specialised classes in a kind-of relationship are not always given individual CRC cards and instead are described in the generalisation class, a *shared card* stereotype (<<shared-card>>) can be applied to show if classes do not have their own CRC card.
- A *common component* stereotype (<<cc>>) is introduced for classes and submodels that are used several times in a model or across models, but maintained centrally.
- Methods are added to the notation to allow a differentiation between constraints and methods (although not shown in the example in Figure 7). Obviously, the concept of methods can be left out of the models if this differentiation is not wanted.

Having defined the PVM notation, the matching CRC card view must be defined. Moving from a PVM to a hierarchical list of classes presents one basic problem, namely that a PVM includes two relationship types, while a hierarchical list includes only one. To include both relationship types in the hierarchical list,

a full line is applied to represent part-of relationships, while a dotted line represents kind-of relationships. This is shown in Figure 8, where the example corresponds to the PVM in Figure 7. It should be noticed that this paper does not deal with a detailed design of CRC cards and that the ones displayed in the figures have only the purpose of illustrating certain aspects.

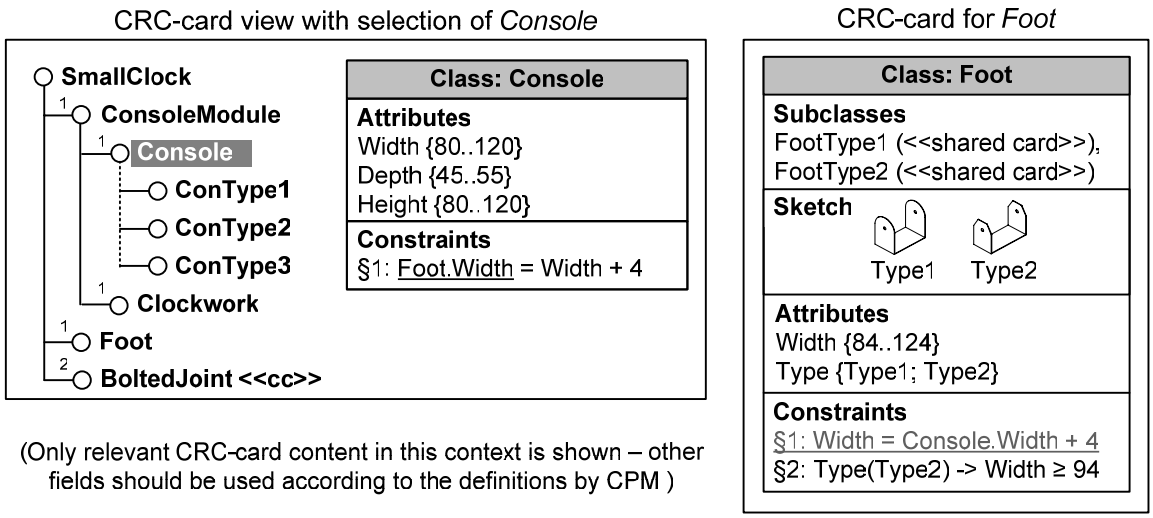


Figure 8: CRC-card view with selection of Console and CRC-card for Foot

In Figure 8, it is seen that the *Console* types are shown as individual classes in the class hierarchy while the *Foot* types are not. This is because the kind-of relation from *Foot* to its subclasses is marked with the shared card stereotype (as seen in Figure 7), contrary to the relation between *Console* and its subclasses.

In Figure 8, the first constraint (§1) is placed on both cards, but with different class references and in grey text on the CRC card for *Foot*. It is intended that this constraint should be created automatically in the CRC card for *Foot*, which is made possible by allowing attributes from external classes to be selected from a list when creating constraints. In the CRC card for *Console* the external class in the constraint (*Foot*) is underlined, as this is intended to be a hyperlink to this class. In the CRC card for *Foot* the constraint, which is owned by *Console*, is also a hyperlink to the CRC card for the class *Console*.

4.4 From PVM to class diagram

Class diagrams have a richer and more flexible notation than PVMs, for example by including more relationship types, allowing multiple inheritance and multiple wholes. On the other hand, PVMs do not include notation that cannot be expressed in class diagrams, which is why defining the transfer from PVM to class diagram is quite straightforward. The only real changes are that subclasses in kind-of relations with the shared card stereotype are not transferred to the class diagram, and sketches do not appear in class diagrams. It would be possible, however, to include sketches within a normative use of UML by placing these inside the notes symbol.

As some might wish to remove or add classes during a transfer from a PVM to a class diagram, it should be possible to select if classes should be excluded or added during the transfer.

4.5 Class diagrams and matching CRC cards

Figure 9 depicts an example that shows the model elements to be included in class diagrams. The example corresponds to the PVM model in Figure 7, but with the addition of an attribute (*ManufacturingTime*) and a method (*CalcManTime*) in the class *Console*, and by adding the classes *AluminiumComp* and *OtherConstraints*.

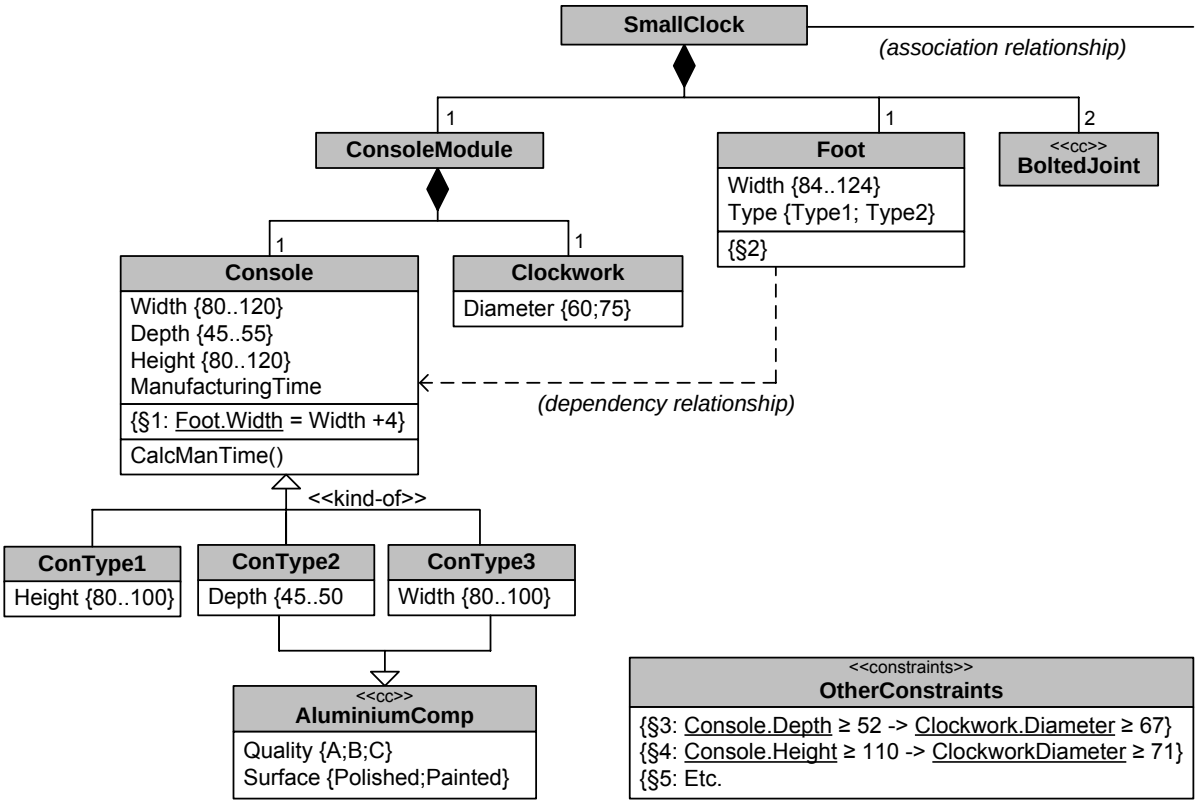


Figure 9: Class diagram definition

Compared to the class diagram definitions of the CPM procedure the following modifications have been made:

- The dependency relationship is added to the selection of model elements used in class diagrams. This relationship type is very useful for illustrating if classes depend on others, *i.e.*, if a class holds a constraint that affects another class, such as between *Console* and *Foot* in Figure 9.
- A *kind-of* stereotype (<<kind-of>>) is added to deal with multiple inheritance in the class hierarchy section of the CRC card view. The subclasses in a generalisation relationship with the kind-of stereotype are shown below the superclass in the main class hierarchy, while other generalisation relationships are shown below the main class hierarchy. If, on the other hand, a class has multiple wholes, the class is shown twice in the main class hierarchy.
- Besides the constraints that are formulated behind attributes, a field for holding constraints is added to the class element. As with PVMs, it can be chosen only to display a constraint identifier (*e.g.*, §2) and state the constraint on the CRC card.
- A *constraints* stereotype (<<constraints>>) is introduced to support the placement of constraints in special classes. Where constraints can be placed in standard PCSs differ, and obviously also where the developers choose to place these. Sometimes constraints are placed in a class (or a similar construct) and sometimes elsewhere.
- As for PVMs, the common component stereotype (<<cc>>) can be applied for classes that are used at several different places in a model or across models, but maintained centrally.

In Figure 10, the CRC card view that matches the class diagram in Figure 9 is shown. In the class hierarchy section, it should be noticed that the dependency relation is not shown, but its presence appears in the matching CRC card. Also, it is seen that the generalisation relation from *Console* to *ConType2* and *ConType3* is shown in the main model while the relation from *AluminiumComp* is shown below the model. This is because the relation from *Console* is marked with the kind-of stereotype as opposed to the

relation from *AluminiumComp*. To incorporate multiplicities into the CRC cards, this information is stated behind relationship classes, as done for *Console* and its superclass *ConsoleModule*.

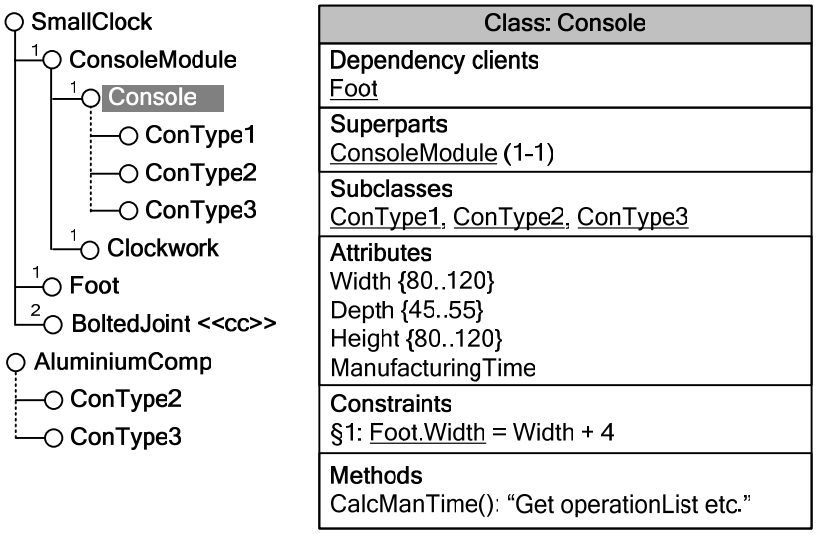


Figure 10: CRC-card view with selection of Console

In Figure 11, the CRC card view with *ConType3* selected is shown. As *ConType3* appears twice in the class hierarchy, both instances are highlighted. It is also seen that the attribute *Width* does not appear to be inherited from *Console*. This is because the inherited attribute has been overridden by defining a new value domain for the attribute.

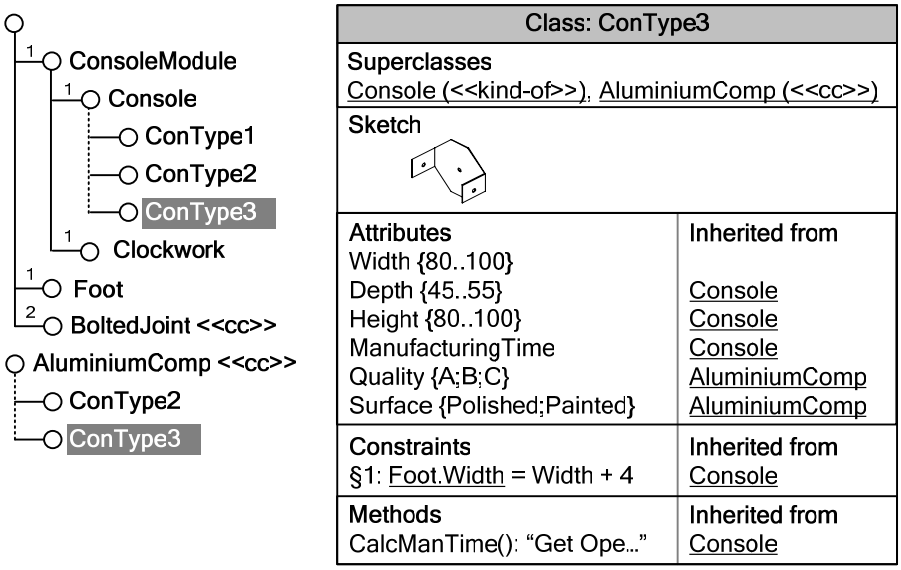


Figure 11: CRC-card view with selection ConType3

4.6 From class diagram to PVM

Contrary to the transfer from PVM to class diagram, the definition of the transfer the other way around is not that straightforward. A basic problem is that class diagrams include more relationship types and allow multiple inheritance and wholes.

Generalisation and composition relationships in class diagrams should be transferred to PVMs. In cases where a class has multiple superclasses in a class diagram, only one of the relationships can be shown in the main class hierarchy of a PVM. Using the same principle as for the class hierarchy section of the CRC

card view, the kind-of stereotype can be applied to show which of the relationships to include in the main class hierarchy of a PVM, while other generalisation relationships are placed below the main model.

The remaining relationship types in the defined selection of class diagram elements are association and dependency. Even though these are not part of the PVM notation, they could in principle be shown on a PVM, though this could make the PVM representation a bit confusing. As a starting point it is chosen not to include the two relationships in the transfer from PVM to class diagram.

The transfer of model elements from a class diagram to a PVM should to a great extent be determined by the selections of the user, *e.g.*, if a specific class should be excluded or added during this operation.

4.7 Interrelated models

A product model can consist of several interrelated submodels, which creates a need to organise models at a higher level. These submodels can be representations of the physical components of a product, but they can also represent different aspects. In Hvam (2004) and Hvam *et al.* (2005b), a framework that defines different kinds of product models is found. The division includes property models, product models and models of meetings with life phase systems, which are all further subdivided into more specific kinds of models. In addition, Hvam *et al.* (2005b) describe three different views of a specific model: customer view, development view and production view.

To deal with grouping of models, the package notation from UML is applied. Figure 12 outlines how package diagrams can be included in the documentation system.

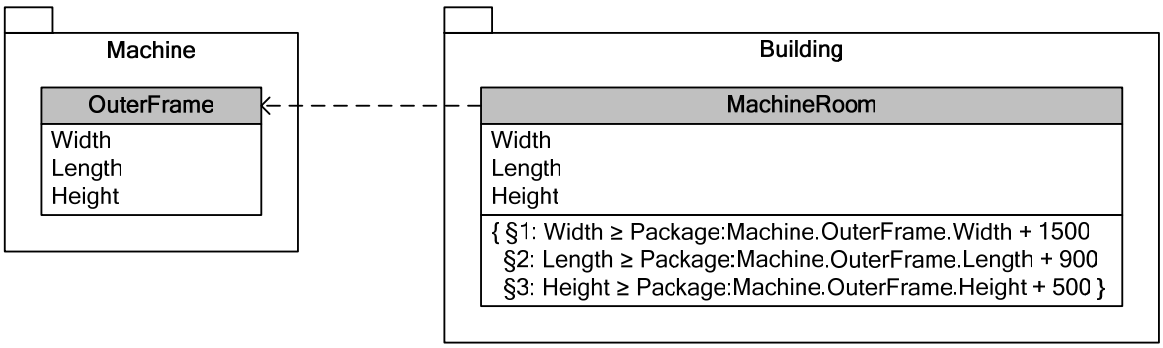


Figure 12: Package model

In Figure 12, the two packages symbolise two product models, which are connected by a dependency relation between classes from each of these. In this simple example it should be imagined that there is a building model, which includes among other classes the class *MachineRoom*. The class *MachineRoom* includes constraints that ensure a machine can fit into the room. The two packages in the example would, as mentioned, consist of other classes, but if the depicted dependency relation is the only thing that connects the two packages, it is adequate to show the two relevant classes.

In Figure 13, a principle of how to navigate between models in the CRC card view is outlined. In this top-level view, the window *Model navigation* can be used to navigate between models, the window *Model structure* can be used to navigate between elements within a model and in the window to the right the matching CRC cards are shown.

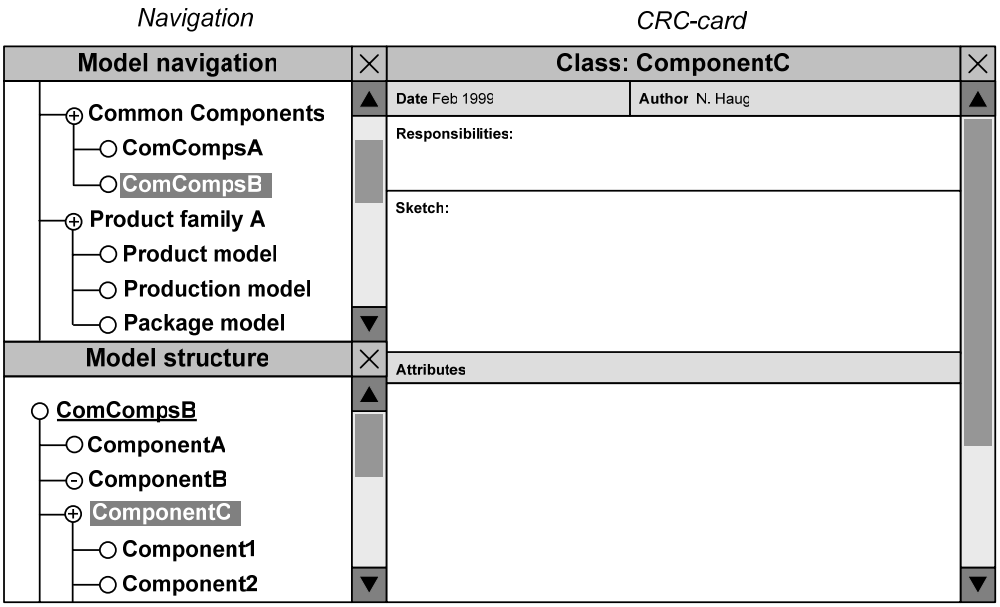


Figure 13: Top level navigation principle

The parts of the product models that are included in the package models should be imported from existing product models and be connected to the package models by the common data model. This means that if an element of a class changes in a product model, this should automatically be reflected in a package model that includes this class.

5 Conclusions

In this paper, a much-needed concept on how to integrate different modelling techniques in a documentation system that supports the CPM procedure for building PCSs was presented.

The fact of not having a documentation system that supports the modelling techniques of the CPM procedure means that different software must be applied throughout a project. It might be chosen to use MS Visio for creating PVMs, Rational Rose for class diagrams and MS Word for CRC cards in the development phase, and finally an application like Lotus Notes in the maintenance phase. This approach implies several manual transfers of information between models without automated checks for consistency across models. A tool that supports the mentioned techniques in an integrated fashion could therefore ease documentation tasks considerably.

A study of earlier research shows that the documentation task is often one of the first to be cut out of a PCS project. Such a decision may later turn out to have very negative consequences, as some companies that did not document their PCS found themselves unable to maintain or further develop their PCS. The choice of not documenting a PCS can be seen as a consequence of the fact that the documentation task is too time consuming. Therefore, it seems fair to assume that a documentation system which in a user-friendly and adequately extensive way supports documentation tasks would lead to greater documentation efforts in companies applying PCSs. This assumption should be seen in the light of the fact that the use of a simple application developed within Lotus Notes, which only supports part of the CPM procedure, despite its limitations, is claimed to have produced significant benefits.

Earlier research regarding how to create a documentation system to support the CPM procedure has mainly dealt with functional requirements. However, definitions of the included techniques and in particular how these should be mapped to each other were not present in an adequate manner prior to this paper. By forwarding these specifications, an important part of the basis for the creation of a documentation system to support the CPM procedure is laid. Much more still needs to be defined regarding issues such as how to handle change requests, and the definition of a common data model together with the creation and tests of prototypes. This work, basing on the definitions presented in this paper, is currently ongoing at CPM.

References

Arlow, J. and Neustadt, I. (2005) *UML 2 and the Unified Process*, 2nd ed., Upper Saddle River, NJ: Addison Wesley.

Beck, K. and Cunningham, W.A. (1989) 'A laboratory for teaching object-oriented thinking', *SIGPLAN Notices*, Vol. 24, No. 10, pp.1–6.

Bennet, S., McRobb, S. and Farmer, R. (2002) Object-Oriented Systems Analysis and Design Using *UML*, 2nd ed., Glasgow: McGraw-Hill.

Edwards, K., Hvam, L., Pedersen, J.L., Møldrup, M. and Møller, N. (2005) Udvikling og implementering af konfigureringssystemer: Økonomi, Teknologi og Organisation *[Development and implementation of configuration systems: Economy, Technology and Organisation]*, Final report from research project, Lyngby: Department of Manufacturing Engineering and Management, Technical University of Denmark.

Felfernig, A., Friedrich, G. and Jannach, D. (2000) 'UML as domain specific language for the construction of knowledge based configurations systems', *International Journal on Software* Engineering and Knowledge Engineering, Vol. 10, No. 4, pp.449–470.

Fowler, M. (2005) *UML Distilled*, 3rd ed., Boston, MA: Addison-Wesley.

Hansen, B., Riis, J. and Hvam, L. (2003) 'Specification process reengineering: concepts and experiences from Danish industry', Proceedings of the 10th ISPE International Conference on Concurrent Engineering: Research and Applications, Madeira, Portugal, 26–30 July.

Harlou, U. (2005) Developing Product Families Based on Architectures: Contribution to a Theory *of Product Families*, Unpublished dissertation, Lyngby: Department of Mechanical Engineering, Technical University of Denmark.

Haug, A. and Hvam, L. (2005) 'Developing 3D configuration systems for manufacturers of complex building components', *Proceedings of IMCM05*, Klagenfurt, Austria, 2–3 June.

Hvam, L. (1994) 'Application of product modelling – seen from a work preparation viewpoint', (Trans), *PhD Thesis*, Lyngby: Department of Industrial Management and Engineering, Technical University of Denmark.

Hvam, L. (2004) 'A multi-perspective approach for the design of product configuration systems – an evaluation of industry applications', Proceedings of International Conference on Economic, Technical and Organisational Aspects of Product Configuration Systems, Lyngby: Department of Manufacturing Engineering and Management, Technical University of Denmark.

Hvam, L. and Malis, M. (2001) 'A knowledge based documentation tool for configuration projects', Proceedings of World Congress on Mass Customization and Personalization, Hong Kong, 1–2 October.

Hvam, L., Pape, S., Jensen, K.L. and Riis, J. (2005a) 'Development and maintenance of product configuration systems – requirements for a documentation tool', *International Journal of Industrial Engineering*, Vol. 12, No. 1, pp.79–88.

Hvam, L., Mortensen, N.H. and Riis, J. (2005b) 'Produktkonfigurering (product configuration)', Publication for Teaching at the Technical University of Denmark, Lyngby: Department of Manufacturing Engineering and Management, Technical University of Denmark.

Hvam, L. and Riis, J. (1999) 'CRC cards for product modeling', *Proceedings of the 4th Annual* International Conference on Industrial Engineering Theory, San Antonio, Texas, 17–20 November.

Hvam, L., Riis, J. and Malis, M. (2002) 'A multi-perspective approach for the design of configuration systems', Proceedings of the 15th European Conference on Artificial *Intelligence*, Lyon, France, 21–26 July.

OMG (2005) *Unified Modeling Language: Superstructure*, Version 2.0: Formal/05-07-04, www.uml.org.

Pulkkinen, A. (2000) ‘A framework for supporting development of configurable product families’, *Proceedings Norddesign*, Lyngby: Technical University of Denmark.

Riis, J. (2003) ‘Fremgangsmåde for opbygning, implementering og vedligeholdelse af produktmodeller – med fokus på konfigureringssystemer (Procedure for building, implementing and maintaining product models – with focus on configuration systems)’, *PhD Thesis*, Lyngby: Department of Manufacturing Engineering and Management, Technical University of Denmark.

Sabin, D. and Weigel, R. (1998) ‘Product configuration frameworks – a survey’, *IEEE Intelligent* Systems and their Applications, Vol. 13, No. 4, pp.42–49.

Stumptner, M. (1997) ‘An overview of knowledge-based configuration’, *AI Communications*, Vol. 10, No. 2, pp.111–125.

Paper C2

CRC-cards for the development and maintenance of product configuration systems

Haug, A. and Hvam, L. (2006): "CRC-cards for the development and maintenance of product configuration systems", in Customer Interaction and Customer Integration (Proceedings of the Joint Conference IMCM'06 & PETO'06, Hamburg, Germany, June 22-23, 2006), GITO-Verlag, Berlin.

CRC-cards for the development and maintenance of product configuration systems

Anders Haug and Lars Hvam

Abstract

This paper presents a new definition of special CRC-cards (Class, Responsibility and Collaboration) to be used in the development and maintenance of product configuration systems (PCS).

CRC-cards are an informal and user-friendly technique for object-oriented modelling. In the early phases of a software development project, CRC-cards can be useful for coming up with design alternatives, as moving the cards around allows exploration of interactions between classes. In 1994, extended CRC-cards with the purpose of holding detailed descriptions of classes in other diagrammatic representations were incorporated into a procedure for developing PCS's. This procedure has since been applied in several configuration projects and further developed at the Centre for Product Modelling (CPM) at the Technical University of Denmark.

The proposed use of CRC-cards by CPM has been a valuable method to support the development and maintenance of PCS's for a number of companies. Investigations of two companies who apply CRC-cards to document the knowledge in their PCS's, however, showed that the contents of their CRC-cards differed from the definitions by CPM in many respects. In this paper these modifications are incorporated into a new definition of CRC-cards, which besides improving the basis for companies taking up the technique, is an important input to the project at CPM concerning the creation of a software-based documentation system that supports the development and maintenance of PCS's.

Keywords:

Product configuration, product modelling, CRC-cards, documentation of product models

1 Introduction

The use of product configuration systems (PCS) is a technology that supports the manufacturing paradigm of mass customisation (Pine et al., 1993). A PCS can be defined as a product-oriented expert system, which allows users to specify products by selecting components and properties under restriction of valid combinations. Applying PCS's can produce benefits such as: shorter lead times, reduction of resources needed to produce specifications and fewer errors in specifications (Hvam, 2004).

The development of a PCS entails a relocation of knowledge from domain experts to a software system. One of the greatest challenges in a configuration project concerns the representation of domain knowledge (Sabin & Weigel, 1998; Hansen et al., 2003). To visualise and capture domain knowledge, various diagrams can be used for creating graphical models. Diagrams for organising elements and their relations (such as class diagrams) can provide a good overview of the contents and structure of a model, but can easily become confusing if too much information is included. To avoid this information overflow, these models can be extended by special CRC-cards (Class, Responsibility and Collaboration), where more detailed information about model elements can be placed (Hvam, 2004).

Experience shows that if a PCS of a certain size is undocumented, it can be difficult or even impossible to maintain (Edwards et al., 2005). CRC-cards can be useful for documenting the knowledge in a PCS, as CRC-cards compared to the modelling environment of many PCS's can hold easily comprehensible descriptions of what is in a PCS. Having such external descriptions can ease the tasks of updating knowledge bases and tracking errors.

Several companies have applied CRC-cards for documenting the knowledge in their PCS's based on definitions of a procedure for developing PCS's from the Centre for Product Modelling (CPM) at the Technical University of Denmark. CPM's CRC-card definition provides an easily understandable basic layout that can be extended by companies according to their individual needs. This does, however, not mean that there is not a need for more extended descriptions of how to apply CRC-cards. The fact of having only a basic definition of CRC-cards can represent a problem if the standard description of how to elaborate CRC-cards is applied uncritically, as this holds risks of having redundant information in the documentation and neglecting to document important aspects.

At CPM it has for some years been an ambition to create a software-based documentation system to support the development and maintenance of PCS's. Documentation systems that support only parts of the CPM-procedure have been created (Hvam & Malis, 2001), but the ambition is to create a system which offers a much more complete support. Therefore, CPM is currently involved in a project of creating such a system, based on definitions by e.g. (Hvam et al., 2005a). For a documentation system to fulfil the needs of different companies, the existing CRC-card definitions have to be extended, e.g. by fields for change management.

This paper presents a new definition of CRC-cards, which besides providing an improved basis for companies who apply CRC-cards in their PCS projects, is an important input to the project at CPM concerning the creation of a software system to support the development and maintenance of PCS's.

This paper is organised as follows. In section 2 the use of CRC-cards in two procedures for developing PCS's is outlined. In section 3 studies two of companies applying CRC-cards to document the knowledge in their PCS's are presented. Next, in section 4 a new definition of CRC-cards to support the development and maintenance of PCS's is presented. The paper ends with a conclusion in section 5.

2 CRC-cards for the development and maintenance of PCS's

The CRC-card technique was invented by Ward Cunningham in the late eighties (Fowler, 2005) and originally presented in (Beck & Cunningham, 1989), where it was described as a way of teaching the object-oriented way of thought. A CRC-card consists of the *class name* together with two columns for *responsibilities* and *collaborators*, as seen in figure 1. In brief, responsibilities are summarisations of the things that an object from the class should do, while collaborators are the other classes with which the class needs to work together (Fowler, 2005).

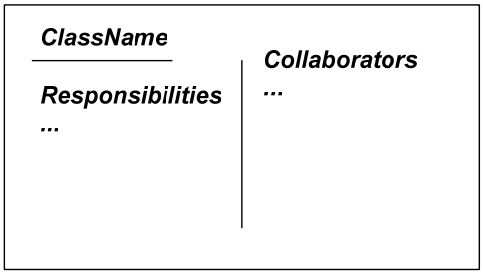


Figure 1: The original CRC- cards (Redrawn from Beck & Cunningham, 1989)

CRC-cards can be a valuable technology for coming up with good object-oriented designs, as moving the class cards around allows exploration of interactions between classes. Although CRC-cards are not part of the Unified Modelling Language (UML), they are a popular technique among skilled object designers (Fowler, 2005). CRC-cards are often used in role-playing sessions, which can be organised according to different principles (Fowler, 2005; Bellin & Simone, 1997; Bennet et al., 2002).

2.1 CRC-cards according to the CPM-procedure

Hvam (1994) presented a procedure for building product configuration systems, which has since been applied in several projects and further developed at CPM. The CPM-procedure consists of the seven phases: 1 Process analysis, 2 Product analysis, 3 Object-oriented analysis, 4 Object-oriented design, 5 Programming, 6 Implementation and 7 Maintenance.

The CPM-procedure proposes the use of three main techniques for modelling the knowledge to be included in a PCS: product variant masters (PVM), UML class diagrams and CRC-cards. In the product analysis phase, PVM's are prescribed for describing the product assortment. Later in the object-oriented phases, the contents of PVM models can be formalised and extended by using class diagrams. The basic argument for including both diagrams is that PVM's seem to be more user-friendly, which can be beneficial in modelling tasks that include domain experts with limited modelling skills. On the other hand, class diagrams are richer and more formalised, which makes these better suited for describing what should be implemented in a PCS.

The purpose of CRC-cards in the CPM-procedure is to hold detailed information about classes represented in PVM's and class diagrams, as illustrated in figure 2. For descriptions of PVM's and class diagrams, according to the definitions of the CPM-procedure, see e.g. (Hvam et al., 2002; Hvam, 2004).

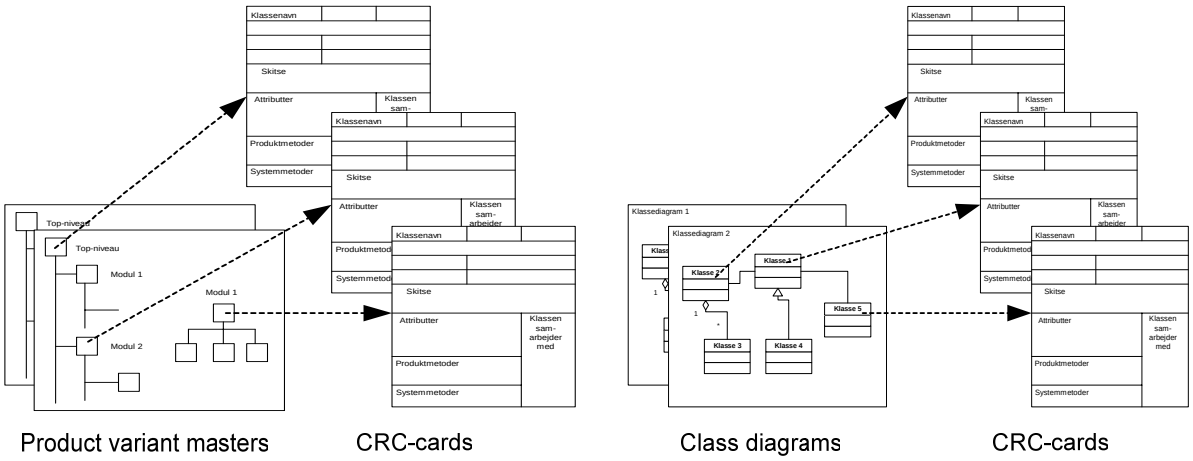


Figure 2: Diagrams connected to CRC-cards (Hvam et al., 2005b)

The CRC-cards defined in (Hvam, 1994) are compared to (Beck & Cunningham, 1989) extended by fields to describe *superclasses/subclasses* (generalisation-specialisation relationships) and *superparts/subparts*

(whole-part relationships). Furthermore, *responsibilities* and *collaborations* are divided into two sections called *Knows* and *Does*, which in UML terminology are basically other names for *attributes* and *methods.*

Later, Hvam and Riis (1999) presented a revised definition of the CRC-cards to be used in PCS projects. Compared to (Hvam, 1994), the proposed definition includes the addition of the fields: *Author*, *Date*, *Sketch* and *Object mission.* This CRC-card definition is seen in figure 3.

Object no.	**Object name**	**Date**	**Author**
Object mission			
Superparts:		**Superclass:**	
Subparts:		**Subclass:**	
Sketch:			
Responsibilities			**Collaborations**
Object knows			
Object does			

Figure 3: CPM CRC-cards (Redrawn from Hvam, 1999)

The CRC-cards were further described by Hvam et al. (2003), where the field *Object mission* is renamed *Responsibilities* and the caption *Responsibilities* to *Know/Does.* In the latest definition from Hvam et al. (2005b) the field *Version* is included, and the fields *Knows* and *Does* are replaced by: *Attributes*, *System methods* and *Product methods*, where the product methods are divided into internal and external. This CRC-card definition is seen in figure 4.

Class name:	Date:	Author/version:
Responsibilities:		
Aggregation	Generalisation	
Superparts:	Superclasses:	
Subparts:	Subclasses:	
Sketch:		
Attributes:	Collaborators:	
System methods:		
Product methods:		
Internal methods: External methods:		

Figure 4: The latest CRC-card definition by CPM (Translated from Hvam et al., 2005b)

On the CRC-card in figure 4, the methods fields are intended to hold information about both methods and what in other contexts is referred to as *constraints* or *(production) rules* (Stumptner, M., 1997; Sabin & Weigel, 1998). *Product methods* concern knowledge about products and their life phase properties, while *system methods* concern software aspects of a PCS. *Internal methods* concern the internal structure, functions and properties of a class, while *external methods* concern interfaces to others classes (Hvam et al., 2005b).

2.2 A procedure for conceptual modelling of product families

Mortensen et al. (2000) propose a five phase procedure for conceptual modelling of product families in configuration projects. The procedure is developed together with Baan Development, a developer of enterprise resource planning and configuration software. The procedure consists of the phases: 1 Identification of configuration task, 2 Identification of product family master plan, 3 Conceptual modelling of product family master plan, 4 Detailed modelling of product family master plan and 5 Modelling of product family in the configuration system.

The mean for describing the product assortment is the PVM technique (in this context called *product variant master plan*). After creating PVM models, the use of a modelling tool developed by Nielsen and Harlau (1999) is prescribed. Using this tool each class of a PVM is described in *Class Description cards* (CD-cards). Later *Class Responsibility cards* (CR-cards) are applied. CR-cards have the same structure as CD-cards, but instead of modelling independently of a PCS, the language from the configuration system is now used. An extract from the CD/CR cards is shown in figure 5.

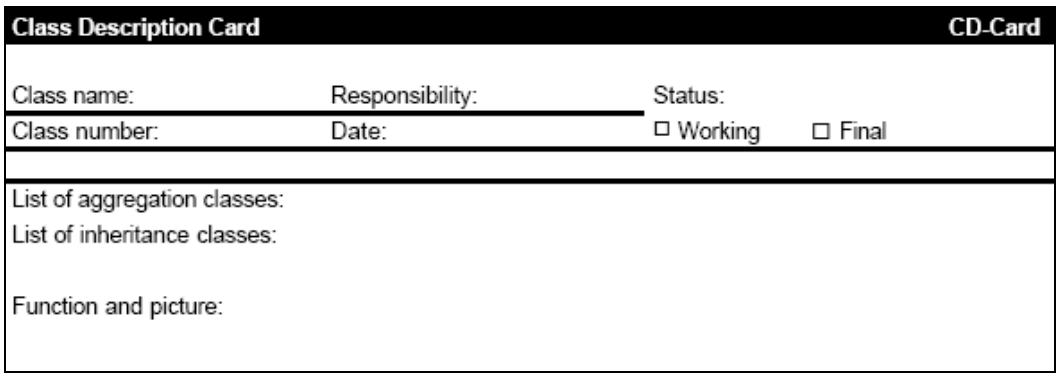


Figure 5: Extract of CD/CR-cards (Mortensen, et al., 2000)

The main contents of the CD/CR-cards are (Mortensen et al., 2000):

- List of aggregation classes: Describes *part-of* relations to other classes.
- List of inheritance classes: Describes the *kind-of* relations to other classes.
- Function and picture: A short description of functionality and a sketch/picture/diagram of the class.
- Defining parameters: Attributes that can be determined directly during configuration.
- Components: Elements, which the class consists of.
- Constraints within the class: Restrictions on how the components and defining parameters may be related.
- Constraints to other classes: Restrictions on how the classes can be related to other classes in the configuration system.
- Mode of action: Description of the input and output parameters for the class. Input and output can be related to the user of the configuration system and other IT systems.
- Sources: Description of the reason for the contents of the CD Card.

3 Case studies

The ways in which the companies GEA Niro A/S and American Power Conversion A/S apply CRC-cards have been investigated by the author. The two companies were chosen because of their relatively structured procedures for using CRC-cards as documentation of the knowledge in their PCS's. The investigations were carried out during 2005 and 2006, through several interviews and by analysing documentation produced by the companies.

3.1 Approach of GEA Niro A/S

GEA Niro A/S (in the following referred to as Niro) is a company with a leading market position within the area of industrial drying technology. Niro is a part of the *GEA Process Engineering Division*, which is represented in more than 50 countries and retains a total of about 3,200 employees. In Denmark, Niro employs more than 400 people.

The products of Niro can be characterised as highly individualised, meaning that much time is used for the creation of product specifications. The PCS project of Niro focuses on the quotation process and aims at: reducing lead times, reducing resources spent on making quotations, optimization of product design and formalisation of product knowledge. At the moment the PCS only supports certain products, but more will be included.

Niro applies CRC-cards for documenting the knowledge in their PCS, which includes thousands of attributes and rules spread across several product families. The CPM-procedure formed the basis for the development of the PCS, but compared to the definitions by CPM, their CRC-cards have been modified to reflect the modelling environment of their standard configuration software (Oracle Configurator) and include fields for management of changes.

Niro had implemented Lotus Notes Release 5 throughout the company as a standard application, wherefore it was chosen to base their documentation on this application, using Notes templates as CRC-cards. Besides CRC-card templates, the Lotus Notes application provides a hierarchical list of the classes, which can be used to navigate between cards. The CRC-card layout with the five folders collapsed is seen in figure 6 (a dummy class).

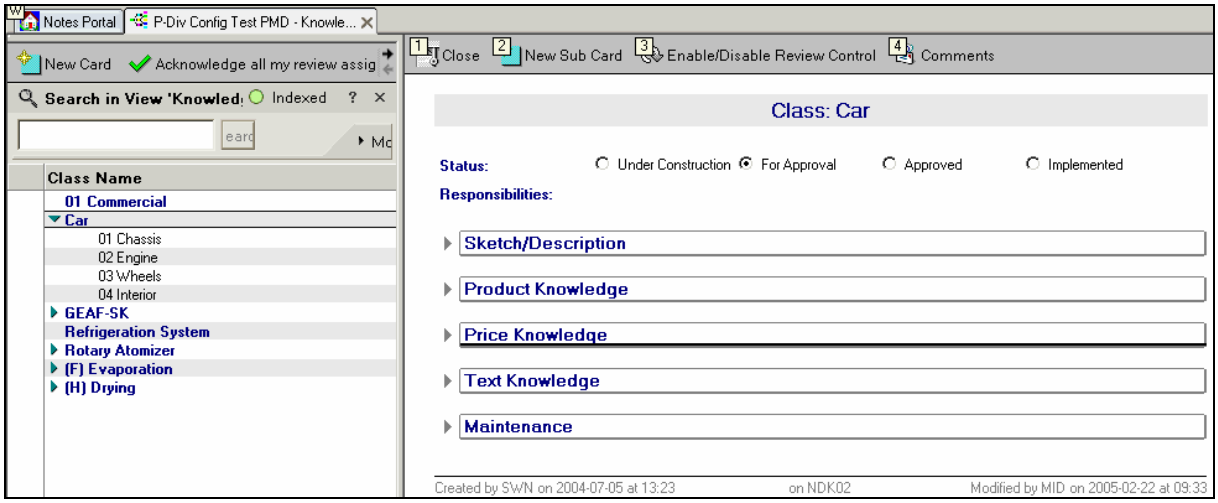


Figure 6: CRC-card layout of Niro

In the CRC-cards of Niro their knowledge is divided into the categories of: *product*, *price* and *text knowledge*. This division means that different kinds of domain experts can access the cards from different places and hereby avoid mixing their information. The three kinds of knowledge are all subdivided into *attributes* and *rules.*

The CRC-cards in the documentation system are connected by hyperlinks. For instance, if an attribute from an external class is part of a rule in a CRC-card, the CRC-card of the external class can be reached by clicking on the specific attribute. The code of the PCS is not described in the CRC-cards, as these are aimed at domain experts and therefore formulated on a higher level of abstraction.

Already implemented cards can be modified and have new elements added. This means that some elements of a CRC-card might be implemented, while others are only requested. To differentiate between elements of different states in the CRC-cards different colours are applied.

The configuration project at Niro is further described in (Hvam, 2004).

3.2 Approach of American Power Conversion A/S

American Power Conversion A/S (APC) produces data centre infrastructure such as uninterruptible power supplies, battery racks, power distribution units, cooling equipment etc. APC has sales offices throughout the world and also manufacturing facilities in several countries, including one in Denmark, where a configuration team of around thirty persons is placed. This team is responsible for the development and maintenance of the PCS's of APC. These PCS's are used worldwide and support the elaboration of quotations and manufacturing specifications. APC's use of configuration technology has led to great benefits, e.g. reductions of lead times from weeks to hours.

The PCS's of APC include thousands of attributes and rules, and are documented in CRC-cards. Like Niro, APC uses Lotus Notes for administering their CRC-cards. Even though the CPM-procedure formed the basis for the development of their PCS's, the CRC-card layout of APC differs much from the CPM definitions. These modifications reflect the products and the modelling environment of the standard configuration software (CinCom Knowledge Builder), and include fields and functionality for management of changes.

The CRC-cards of APC fall into three categories: *product* (product descriptions), *lifecycle* (descriptions of the lifecycle of the product) and *specification* (descriptions of the documents throughout the life phases of products). Besides using CRC-cards for describing product related knowledge, APC also applies CRC-cards for describing requested changes of existing cards. Who can change a CRC-card depends on the state of the card and the rights of the user.

The CRC-cards of APC are primarily used by domain experts, wherefore the code from the PCS's is not described in the CRC-cards. Instead, this knowledge is formulated on a higher level of abstraction.

The CRC-cards contain the following top level information: *Title*, *Version*, *Category*, *Sub-Category*, *Requested Go Live Date* and *Phase*. The CRC-cards further contain the fields:

- Class Description
- Domain Expert (Product responsible)
- Link to other CRC-cards
- Sketch/Picture
- Knows (i.e. attributes) and Collaborations
- Rules and Collaborations (Rules are divided into configuration and selection rules)
- SKU's/Parts (Variants and subparts with and without product codes)
- Edit History (Who created different versions of the card)

From the CRC-cards it is possible to: request acknowledgement from a domain expert, search for CRC-cards linked to the current solution and get help to fill out the CRC-card.

The configuration project at APC is further described in (Hvam, 2004).

4 Definition of CRC-cards for the development and maintenance of PCS's

4.1 Discussion of existing designs versus empirical investigations

In spite the fact that the CPM-procedure, as mentioned, formed the basis of the PCS projects at both Niro and APC, their CRC-card layouts differ much from the CPM definitions.

In the newest CRC-card definition from CPM, methods (in this context including rules/constraints) are divided into system methods and product methods. This division does not correspond with either of the investigated companies, as they do not include the system aspect. The distinction of CPM, however, could be useful in other cases.

Besides not dividing and naming their *knowledge* according to the CPM definitions, Niro also includes a status-field that tells if a card is: *under construction*, *waiting for approval*, *approved* or *implemented*. When using CRC-cards for documentation in the maintenance phase, it is in most cases necessary to include this type of information to be able to administer changes.

The CRC-card template of APC also differs much from the CPM definitions. Besides naming and dividing attributes and rules differently, the cards of APC: are categorised, include a requested go-live-date and have a field for stating the owner of the real product together with a functionality that allows requesting acknowledgement from this person. Like the CRC-cards of Niro, the ones of APC also include a status-field, in this case called *Phase*. Furthermore, the CRC-cards of APC include fields that reflect that the CRC-cards, besides being used for describing classes, are also used for describing change requests.

The CRC-card definition of the procedure by Mortensen et al. (2000) to some degree resembles the CPM definition, but is more loosely connected to class diagrams and closer to the Baan configurator software. This is for instance seen from that the term *parameters* is used instead of *attributes* and a field for *methods* is not included. But the main difference is the use of two types of cards, CD-cards for describing domain knowledge independently of a PCS and CR-cards for describing what should be implemented in a PCS. In the CRC-card definition provided in this paper, the primary intention of the CRC-cards is that these are used for describing the knowledge that should be (or is) implemented in a PCS, but in a manner which is comprehensible to relevant domain experts, i.e. not PCS code. This purpose is in accordance with the CPM definitions and the way the two investigated companies apply CRC-cards.

Based on the investigations of only two companies it seems to be a hopeless mission to define a standard CRC-card layout that will fit the needs of all companies without these making individual modifications. The aim of the following definitions of a new CRC-card layout is therefore to provide a basis, which to a greater degree is prepared for different kinds of use than the existing definitions.

4.2 The basic layout of the new CRC-cards

The proposed new CRC-card layout consists of some top level information together with six types of folders for class information. A one-page layout with expandable/collapsible folders is chosen, as this, contrary to a multi-page layout, allows information from any two folders to be shown on the screen simultaneously. The proposed layout with the folders collapsed is shown in figure 7.

Class:	Status:	Change requests:	Version:
▶ Basic information			
▶ Relationships			
▶ Sketch/Picture			
▶ Knowledge group 1			
⋮ ▶ Knowledge group N			
▶ Change requests			
▶ Change history			

Figure 7: Basic layout

In the top level section, the field *Change requests* is intended as a field which tells whether there are changes that need to be implemented in the current card, i.e. *yes* or *no*. A card can therefore have the status *implemented*, even though later requests for changes have emerged.

It should be noticed that the folders called *Knowledge group 1-N* are intended to be applied for grouping of different kinds of knowledge and to be named according to preferences in individual projects.

4.3 Basic information

The *Basic information* folder holds information about: the creator of a CRC-card, who is responsible for the card, and who is responsible for the product or component that is represented by the class. In this folder also the main responsibilities of the current class can be described. In figure 8 the fields within the *Basic information* folder are shown.

Class:		Status:	Change requests:	Version:
▷ Basic information				
Created by:	Date:	Card responsible:	Product responsible:	
Responsibilities:				
▶ Relationships				
▶ Sketch/Picture				
▶ Knowledge group 1				
⋮ ▶ Knowledge group N				
▶ Change requests				
▶ Change history				

Figure 8: Basic information

Besides the fields shown in figure 8, it could in some cases be relevant to include fields that describe system interfaces.

4.4 Relationship types

The CPM-defined CRC-cards only include information about two relationship types, *generalisation* and *aggregation*, where, in this context, the latter is used without differentiation between the two types of aggregation, *aggregation* and *composition* (or *composite aggregation*). The difference between the two types of aggregation is that the composition relationship requires that a part instance is included in at most one composite at a time, and that the composite object is responsible for the creation and destruction of the part (OMG, 2005; Fowler, 2005). The properties of a composition relationship therefore correspond better to the perception of a product composed of parts, wherefore this relationship type is used in this paper.

A class diagram can also include other relationship types. In the CPM-procedure (Hvam et al., 2005b) a third kind of relationship is applied, namely *associations*. Consequently, this relationship type is included in the new CRC-card layout. Also the dependency relationship is included in the new definitions, as this can be useful for displaying that a constraint in one class affects another class. For a complete description of the relationships in class diagrams see (OMG, 2005).

Based on these observations, a basic layout for the *Relationships* folder is defined, as shown in figure 9. Here multiplicities can be stated in brackets following relationship classes.

▷ Relationships		
Composition	Superparts:	Subparts:
Generalisation	Superclasses:	Subclasses:
Dependency	Sources:	Clients:
Association	Classes:	

Figure 9: Relationships

In cases where PVM's are used for final documentation instead of class diagrams, obviously the dependency and association relationships should be left out, and instead the constraint relationship of PVM's can be added if this is used.

Felfernig et al. (2000; 2001) provide an approach for developing PCS, which does not include CRC-cards but includes class diagrams. In this approach the four mentioned relationships are applied, which are additionally extended by the UML concept *stereotypes*. If CRC-cards are applied together with class diagrams, including stereotyped relationships, then either the stereotype should be stated behind the class names in the relationship fields or the layout shown in figure 9 should be extended with additional fields.

If PVM's or class diagrams are applied together with the CRC-cards, and these diagrams describe all relationships, it would in some cases not be necessary to describe these relationships in the CRC-cards as well. However, stating relationships in the CRC-cards opens the possibility of navigating between CRC-cards without using the two structural diagrams, which could be particularly advantageous in cases with software-supported handling of CRC-cards where these are connected by hyperlinks.

4.5 Sketch/Picture

An example of the use of the field *Sketch/Picture* is shown in figure 10. Here topological variance of the subclasses is shown to the left and the placement of different measure attributes to the right.

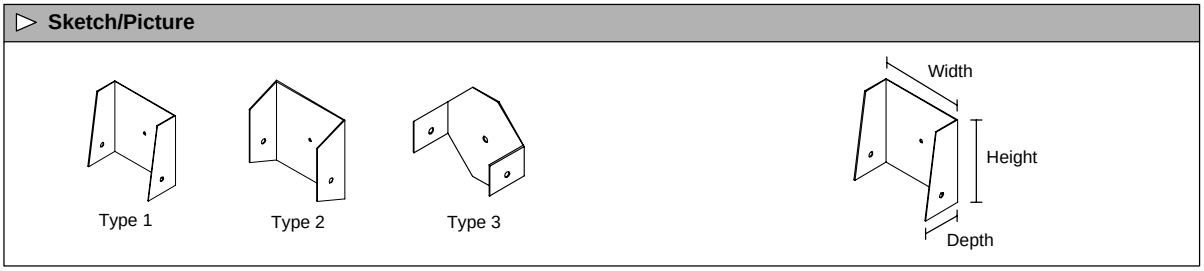


Figure 10: Sketch/Picture

4.6 Knowledge groups

In the CPM definitions and the conducted company studies there are different ways of dividing and naming: rules, constraints, methods and attributes (from now on, referred to as *class knowledge*):

- CPM definitions: Methods are divided into *system methods* and *product methods*, the latter into *internal* and *external*.
- Niro: Knowledge is grouped into *product*, *price* and *text knowledge*, which are all subdivided into *attributes* and *rules*.
- APC: Rules are divided into *configuration* and *selection rules*, and parts divided into parts with and without SKU number when describing product relationships.

The three ways of handling class knowledge differ in two respects, namely the categorisations and the subdivisions of class knowledge.

If all the defined categories of class knowledge from the two companies and the CPM definitions should be included in the CRC-card layout, this would in most cases mean that only few of these categories would be used. It therefore seems that a generic concept (as illustrated in figure 7), where this information is not pre-grouped, but can be defined by a specific company, would be a better solution. Of possible categories of class knowledge can be mentioned: product, process, visualisation, price, text, software etc.

The other question is how class knowledge should be subdivided. On the cards of Niro and APC, the different types of knowledge are divided into attributes and rules, and in the CPM definitions into attributes and methods. In the definitions provided by this paper, concepts from UML are applied in order to conform to a widespread standard. In UML, methods and constraints are not the same thing, and the concept of *rules* is not applied. Here a constraint is an assertion indicating that a condition or restriction must be satisfied by a correct design, which can be expressed by using any formalism, as long as placed within braces ({}). A method is defined as the implementation of an operation, which is a behavioural feature that specifies the name, type, parameters, and constraints for invoking an associated behaviour (OMG, 2005). For instance, an operation that retrieves data from another system would be a method, while an expression that restricts a combination of two specific components would be a constraint.

In the proposed CRC-card layout, a knowledge group is therefore divided into: attributes constraints and methods. In cases where other concepts or terms are used in the modelling environment of a PCS, such as *parameters*, *rules* or *calculations*, the fields can be changed according to preferences.

Based on the chosen definitions the basic content of a knowledge group is shown in figure 11.

Knowledge group 1	
Attributes	
Constraints	
Methods	

Figure 11: Knowledge group

The layout in figure 11 basically lets the users formulate any kind of information within the fields. However, it might in some cases be better to specify what information is needed to ensure that adequate

and relevant information is documented. Therefore, the three fields in figure 11 can be further divided, as shown in the example in figure 12.

In figure 12 it should be noticed that attributes from other classes (collaborators) are qualified by the class and underlined, as they are intended to be hyperlinks in a software-supported environment. This makes the field *Collaborators* from the existing CRC-card layout superficial, unless this is used for other purposes than describing collaborating classes.

▷ Product knowledge						
Attributes	***Name***	***Values***	***Unit***	***Type***	***Inherited***	***Comments***
	Height	30; 35; 40	mm	[Integer]		
	Width	30; 40; 50	mm	[Integer]		
Constraints	***Name***	***Description***			***Inherited***	***Comments***
	§1	Heigth ≥ Chair.Height + 300				
	§2	Type2 -> Legs.Type1 or Legs.Type3				
Methods	***Name***	***Description***			***Inherited***	***Comments***
	CalcArea ()	Height * Width				

Figure 12: Product knowledge

As seen in figure 12, the field *Inherited* is included to tell if an attribute, constraint or method is originating from another class, which can be useful information when making changes.

Besides the fields shown in figure 12, other fields could in some contexts be relevant, such as:

- Implemented (Describing the code implemented in the PCS)
- External systems collaborations (External interfaced systems, e.g. an ERP-system)
- Name in external system (Name of the field in an interfaced system)

In the mentioned approach by Felfernig et al (2001) four class stereotypes are defined: *component*, *resource*, *function* and *port*. Using these stereotypes would, if the logic of creating one CRC-card per class was obeyed, imply that some CRC-cards with sparse information are created. To avoid having more cards than necessary, it might be considered to place this kind of information within the product component classes, of which these elements are a part.

4.7 Handling of change requests

The CPM definition of CRC-cards does not include fields for management of changes, besides a version number. However, the two investigated companies do. As mentioned, Niro uses a different colour of text to indicate a requested change, while APC creates a new CRC-card, which acts as a change request. While the Niro approach seems to include some risk of errors, the APC approach on the other hand seems a bit elaborate. A new concept for handling changes in CRC-cards is therefore defined in the following.

Until now, the CRC-card definition proposed in this paper does not require real software development and could more or less be handled by e.g. MS Word templates. The proposed concept for dealing with change requests, however, requires some software development in order to function efficiently. The basic idea is that a user can click on a line or box on a CRC-card that is to be changed and then select an action: *modify*, *delete* or *add*. After this a change request, where additional change information can be stated, should automatically be created. The three kinds of change requests are illustrated in an example in figure 13, where it should obviously be possible to sort and filter these.

▷ Change requests									
Change ID	**Version**	**Requested by**	**Date**	**Action**	**Status**	**Assigned to**	**Completed by**	**Date**	**Folder**
1245		W. Haug	11/11 05	Modify	Pending	K. Larsen			Product knowledge
Exist. line	**Constraints**	§2	Type2 -> Legs.Type1 or Legs.Type3						
New line	**Constraints**	§2	Type2 -> Legs.Type2 or Legs.Type3						
Change ID	**Version**	**Requested by**	**Date**	**Action**	**Status**	**Assigned to**	**Completed by**	**Date**	**Folder**
1246	V.1.1	W. Haug	14/11 05	Delete	Implem.	K. Larsen	V. Jensen	15/11 05	Product knowledge
Exist. line	**Methods**	CalcArea ()	Height * Width						
Change ID	**Version**	**Requested by**	**Date**	**Action**	**Status**	**Assigned to**	**Completed by**	**Date**	**Folder**
1247	V.1.2	W. Haug	17/11 05	Add	Implem.	K. Larsen	K. Larsen	18/11 05	Product knowledge
New line	**Methods**	CalcVol ()	Height * Width * Depth						

Figure 13: Change requests

4.8 Change history

Based on the concept for handling change requests, the folder *Change history* is defined as a list of change records together with the history of versions. This is shown in figure 14.

▷ Change history									
▷ Versioning									
Version	**Created by**	**Date**	**Comments**						
V.1.0	W. Haug	1/11 05							
V.1.1	V. Jensen	15/11 05							
V.1.2	K. Larsen	18/11 05							
▷ Changes									
Change ID	**Version**	**Requested by**	**Date**	**Action**	**Status**	**Assigned to**	**Completed by**	**Date**	**Folder**
1245		W. Haug	11/11 05	Modify	Pending	K. Larsen			Product knowledge
1246	V.1.1	W. Haug	14/11 05	Delete	Implem.	K. Larsen	V. Jensen	15/11 05	Product knowledge
1247	V.1.2	W. Haug	17/11 05	Add	Implem.	K. Larsen	K. Larsen	18/11 05	Product knowledge

Figure 14: Change history

5 Conclusions

In this paper a new definition of specialised CRC-cards to support the development and maintenance of PCS's was presented.

Empirical studies of two companies showed that CRC-cards can be a valuable technique to support the development and maintenance of PCS's. The studies also showed that the current definitions of CRC-cards in many respects differed from the way in which CRC-cards were applied in praxis. The experience gained was incorporated into a new definition of a CRC-card layout to be used for the development and maintenance of PCS's.

The basic layout of the proposed CRC-card consists of top level information together with six groups of class information. One of these groups, called *Knowledge group*, is to be renamed and have multiple instances according to the preferences of a specific company. For instance, the group would in the case of one of the investigated companies have the instances: *Product knowledge*, *Price knowledge* and *Text knowledge*. The new CRC-card definition hereby offers a flexible basis that to a great extent supports the divisions of information of the CRC-cards, which are applied by the two investigated companies.

Compared to the existing CPM definitions of CRC-cards, the new layout proposed includes several new fields, e.g. for documenting additional relationship types and organising information about attributes, constraints and methods. Another main extension of the CRC-card layout concerns the handling of change requests. Contrary to the other content of the new CRC-card layout, this part requires some software development in order to function efficiently.

By incorporating the experience gained from two companies into a new definition of CRC-cards to support the development and maintenance of PCS's, an improved basis for other companies taking up the technique has been provided. The use of the new definitions could produce possible benefits such as

minimising redundant information and avoiding neglecting relevant aspects in the PCS documentation. The definition of CRC-cards provided in this paper also forms an improved basis for realising the ambition of CPM to create a documentation system to support the development and maintenance of PCS's.

References

Beck, K. & Cunningham, W.A. (1989): A laboratory for teaching object-oriented thinking. *SIGPLAN Notices*, 24 (10), 1-6.

Bellin, D. & Simone, S.S. (1997): *The CRC Card Book*. Reading, MA: Addison-Wesley, 1997.

Bennet, S., McRobb, S. & Farmer, R. (2002): *Object-Oriented Systems Analysis and Design using UML* (2nd edition). Glasgow: McGraw-Hill, 2002.

Edwards, K., Hvam, L., Pedersen, J.L., Møldrup, M. & Møller, N. (2005): *Udvikling og implementering af konfigureringssystemer: Økonomi, Teknologi og Organisation.* [Development and implementation of configuration systems: Economy, Technology and Organisation]. Final report from research project, Lyngby: Department of Manufacturing Engineering and Management, Technical University of Denmark, 2005.

Felfernig, A., Friedrich, G. & Jannach, D. (2000): UML as domain specific language for the construction of knowledge based configurations systems. *International Journal on Software Engineering and Knowledge Engineering*, 10(4), 449-470.

Felfernig, A., Friedrich, G. & Jannach, D. (2001): Conceptual modeling for configuration of mass-customizable products. *Artificial Intelligence in Engineering*, 15(2), 165-176.

Fowler, M. (2005): *UML Distilled* (3rd edition). Boston, MA: Addison-Wesley, 2005.

Hansen, B., Riis, J. & Hvam, L. (2003): Specification process reengineering: concepts and experiences from Danish industry. *Proceedings of the 10th ISPE international Conference on Concurrent Engineering: Research and Applications*, Madeira, Portugal, July 26-30, 2003.

Hvam, L. (1994): *Application of product modelling – seen from a work preparation viewpoint* (Trans). PhD thesis, Lyngby: Department of Industrial Management and Engineering, Technical University of Denmark, 1994.

Hvam, L. (2004): A Multi-perspective approach for the design of Product Configuration Systems – An evaluation of industry applications. *Proceedings of International Conference on Economic, Technical and Organisational aspects of Product Configuration Systems*, Lyngby: Department of Manufacturing Engineering and Management, Technical University of Denmark, 2004.

Hvam, L. & Malis, M. (2001): A Knowledge Based Documentation Tool for Configuration Projects. *Proceedings of World Congress on Mass Customization and Personalization*, Hong Kong, Oct. 1-2, 2001.

Hvam, L., Mortensen, N.H. & Riis, J. (2005b): *Produktkonfigurering*. [Product configuration]. Publication for teaching at the Technical University of Denmark, Lyngby: Department of Manufacturing Engineering and Management, Technical University of Denmark, 2005.

Hvam L., Pape, S., Jensen, K.L. & Riis, J. (2005a): Development and maintenance of product configuration systems - Requirements for a documentation tool. *International Journal of Industrial Engineering*, 12 (1), 79-88.

Hvam L. & Riis J. (1999): CRC cards for product modeling. *Proceeding of the 4th Annual International Conference on Industrial Engineering Theory*, San Antonio, Texas, Nov. 17-20, 1999.

Hvam, L., Riis, J. & Hansen, B.L. (2003): CRC-Cards for Product Modelling. *Computers in Industry*, 50(1), 57-70.

Hvam, L., Riis, J. & Malis, M. (2002): A multi-perspective approach for the design of configuration systems. *Proceedings of the 15th European Conference on Artificial Intelligence*, Lyon, France, July 21-26, 2002.

Mortensen, N.H., Yu, B., Skovgaard, H. & Harlou, U. (2000): Conceptual modeling of product families in configuration projects. *Papers from the Workshop at the 14th European Conference on Artificial Intelligence*, Berlin, Germany, Aug. 21-22, 2000.

Nielsen, M.P. & Harlou, U. (1999): *Modelling Product Families in Configuration Systems*. M.Sc.-thesis, Lyngby: Department of Control and Engineering Design, Technical University of Denmark.

OMG (2005): *Unified Modeling Language: Superstructure* (Version 2.0: Formal/05-07-04). www.uml.org, 2005.

Pine, B.J., Victor, B. & Boynton, A.C. (1993): Making Mass Customization Work. *Harvard Business Review*, 71(5), 108-119.

Sabin, D. & Weigel, R. (1998): Product Configuration Frameworks - A survey. *IEEE Intelligent Systems & Their Applications*, 13(4), 42-49.

Stumptner, M. (1997): An overview of knowledge-based configuration, *AI Communications*, 10(2), 111-125.

Paper C3

Creating a documentation system to support the development and maintenance of product configuration systems

Haug, A., Degn, A., Poulsen, B., and Hvam, L. (2007): "Creating a documentation system to support the development and maintenance of product configuration systems", in Proceedings of the 2007 WSEAS International Conference on Computer Engineering and Applications, Queensland, Australia, Jan. 17-19, 2007.

Creating a documentation system to support the development and maintenance of product configuration systems

Anders Haug, Anders Degn, Bjarne Poulsen, Lars Hvam

Abstract: A product configuration system (PCS) can be defined as a product-oriented expert system that allows users to specify a product while restricting how different elements and properties may be combined. The use of configuration technology has in several cases led to improvements of product specification processes, such as shorter lead times, reductions of resources needed, and fewer errors.

A procedure for building product configuration systems from the Centre for Product Modelling at the Technical University of Denmark has been applied in projects for more than ten years. The CPM-procedure includes three main modelling techniques to support the development and maintenance of PCSs. However, no software, which supports all three techniques in an integrated fashion, currently exists. This means that when developing PCSs based on the CPM-procedure there is no automatic integration between the created models, wherefore some information has to be transferred between models manually. CPM has, therefore, for some years worked on creating a basis for developing a documentation system that supports the development and maintenance of PCSs. Research focusing on the requirements for a documentation system has been produced, and more recently detailed definitions of the included modelling techniques have emerged. This paper describes how these definitions have been converted into a software prototype and what have been learned from the evaluation of the prototype.

Keywords: Product configuration, Knowledge engineering, Documentation of product models, Class diagrams, Product variant master, CRC-cards

1 Introduction

The use of configuration systems has for a number of years been a successful application of artificial intelligence techniques [1]. A product configuration system (PCS) can be defined as a product-oriented expert system, which by applying knowledge of a domain, lets the user specify a product under restriction of valid combinations of product components and properties. In many cases, the application of configuration technology has led to a range of improvements, such as reductions in lead times, number of errors, and use of resources in the specification of products [2, 3, 4, 5, 6].

The representation of domain knowledge is often one of the greatest tasks in a PCS project [7, 8, 9]. In a PCS project knowledge is often represented in two distinctive kinds of models, namely analysis and design models. The models that describe the knowledge of a domain (in this context product knowledge) without considering implementation aspects are called analysis models (or domain models and conceptual models). When creating analysis models, besides describing the knowledge of a domain, also a need for defining new knowledge can exist, as when a PCS project is started, sometimes the included product families do not have a suitable architecture to form a basis for the creation of a generic product knowledge model. The creation of analysis models can therefore require great involvement of the relevant domain experts of a company. Based on the analysis models, design models can be created in order to facilitate the implementation of the domain knowledge in the PCS. Design models can be seen as formalised and possibly further detailed versions of analysis models that take implantation issues into consideration. This means that the requirements for the analysis and the design language are different, in that the analysis language has to be adequately simple in order to be understood by the domain experts involved, while the design language has to be adequately rich and formalised in order to make accurate descriptions of what is to be implemented.

In many cases where manufacturers of customised industrial products apply configuration technology, the knowledge base of the PCS consists of thousands of classes, attributes, constraints, and methods. When a product changes, the knowledge base of a PCS has to be updated in order for the PCS to support the company processes. As the modelling environment of a PCS seldom provides an adequately comprehensible overview for the domain experts to understand the current model that is to be changed, these kinds of updates often require external documentation. Unless the models created as a basis for the development of the PCS have been maintained, these have to be updated or new models created.

The mentioned aspects present a demand for a coherent documentation system that supports the applied modelling techniques in order to avoid tasks such as: manual transfers of information between models, reconstruction of models, and ensuring consistency across models.

Based on an often applied procedure for the development of PCSs from the Centre for Product Modelling (CPM) at the Technical University of Denmark, research on how to create a documentation system has been carried out. However, so far this research has not resulted in systems that include all the three main modelling techniques of the CPM-procedure. This paper describes the creation and evaluation of the first prototype capable of supporting these three techniques in an integrated manner.

The rest of this paper is structured as follows: Firstly, in section 2, the mentioned procedure for the development of PCSs is outlined with a focus on its techniques for the creation of analysis and design models. Next, in section 3, research concerning a documentation system to support the development and maintenance of PCSs is resumed. In section 4 a prototype developed on the basis of the research carried out is described. Section 5 describes the evaluation of the prototype. The paper ends with a conclusion in section 6.

2 The CPM procedure

To carry out a PCS project is often a great task, both in relation to the change of business processes and in relation to the creation of the PCS itself. To support these tasks, in 1994 Hvam [10] presented a procedure for the development and maintenance of PCSs. This procedure has since continuously been updated at CPM, as experience with the procedure has been obtained. The CPM-procedure or part of the procedure has been applied in a number or cases in Danish industry [2, 3]. The CPM-procedure in its current form consists of seven phases to support the course of a PCS project: 1 Process analysis, 2 Product analysis, 3 Object-oriented analysis, 4 Object-oriented design, 5 Programming, 6

Implementation, and 7 Maintenance.

The CPM-procedure prescribes the use of three major techniques for the creation of analysis and design models, where product variant masters (PVM) are prescribed for the creation of analysis models, class diagrams for the creation of design models, and CRC-cards (Class, Responsibility and Collaboration) for making detailed descriptions of the classes in PVMs and class diagrams. However, in some cases PVMs provide an adequate basis for implementation of the product knowledge into a PCS, which, obviously, means that class diagram models do not need to be elaborated [3].

2.1 PVMs

A PVM is a diagram that is applied for describing generic product models. A PVM consists of two sections that describe part-of structure and kind-of structure of a product model. These two structures, in object-oriented terms, roughly correspond to aggregation and generalisation respectively. Class descriptions, attributes, and constraints can be stated below the classes in a PVM. In figure 1 the latest definition from CPM of the notation formalism of a PVM is shown.

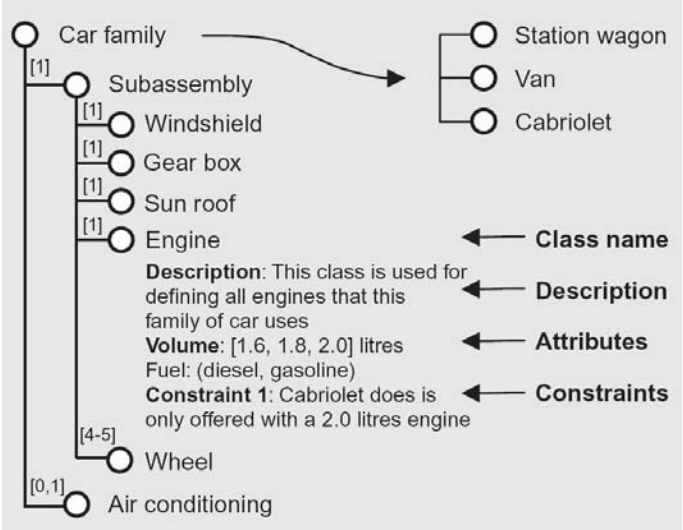


Fig.1. The most recent definition of the PVM formalism [3] from [11]

PVM models are intended to be discussed and refined in repeated sessions of knowledge engineers and domain experts until the descriptions of the product assortment are adequately extensive and detailed. The experience with the use of PVMs in configuration projects is that PVMs can be a strong tool for discussing and defining the product assortment [2]. As no software distinctly aimed at the creation of PVMs exists, PVMs are normally elaborated by the use of programs such as MS Excel or MS Visio.

2.2 Class diagrams

The Unified Modelling Language (UML) is a modelling language with a formal syntax that is defined by the Object Management Group (OMG). UML 2.0 defines thirteen types of diagrams, where one of these is the class diagram [12]. Class diagrams are used for describing the objects in a system together with the various kinds of static relationships among them [13]. Compared to PVMs, class diagrams are far more widespread in use and have a much broader range of application. The notation for the class element and the most common relationship types are shown in figure 2, where a *navigability arrow* can be used to show the direction of association, aggregation, and composition relationships. For more detailed descriptions of class diagrams, see [12, 13].

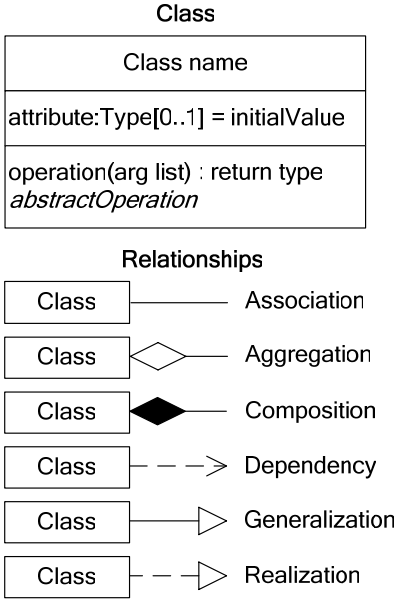


Fig.2. Commonly applied class diagram elements

The CPM procedure prescribes the use of a sub-selection of the relationship types of class diagrams, including generalisation, aggregation, and association [3]. The argument of the CPM-procedure for including both PVMs and class diagrams, which basically are two ways of illustrating the same thing, is that PVMs seem to be more easily understood by domain experts with limited modelling prerequisites, while class diagrams provide a much richer and more formal language, which makes these better suited for the creation of design models [3].

2.3 CRC-cards

The CPM-procedure proposes the use of CRC-cards for holding detailed descriptions of classes in PVMs and class diagrams in order to enhance the clarity of such models. For this purpose, CPM has provided a definition of special CRC-cards to be used in PCS projects. While the original CRC-cards by Beck and Cunningham [14] only include the class name together with two columns for responsibilities and collaborators, the CRC-cards of CPM have been extended with fields for stating: author, date, superparts, subparts, superclasses, subclasses, attributes, and more [2, 3].

3 Towards a documentation system

As mentioned, the CPM-procedure prescribes the use of PVMs, class diagrams, and CRC-cards during the development and maintenance of PCSs. As there is no software available, which supports all three techniques in an integrated fashion, different kinds of software are applied. This implies a need for manual transfers of information between PVMs, class diagrams and CRC-cards, which are time consuming and hold risks of errors. Documentation of the knowledge base of PCSs is therefore an aspect of product configuration that has been subject to some investigations.

The experience with product configuration in twelve Danish companies was investigated in a research project in 2003 and 2004 [15]. The investigations showed that the documentation task was often the first to be given a lower priority or even cut out of the project. It was also pointed out that a lack of documentation of the PCS knowledge base can have very negative consequences, as some companies were unable to further develop their PCSs. Another effect of the lack of documentation was a negative impact on the daily communication between people involved in product development and PCS development respectively.

As models grow in extent and complexity, the effort required for the creation of documentation increases. It is the general impression of Hvam et al. [16] that many companies settle for less

comprehensive documentation than what is actually needed in order to be able to further develop a PCS. They further argue that this to some extend could be avoided by the presence of a documentation system that supports the maintenance of PCS documentation, which would relieve the companies from the time-consuming task of ensuring consistent product models.

3.1 Requirements for the system

As a response to the apparent need for more advanced software for the documentation of the knowledge base of PCSs, research on this topic has been carried out. Based on a survey of five standard configuration systems and the experience gained from several Danish configuration projects, Hvam and Malis [17] define the requirements for a documentation system: easy to maintain, facilitates the modelling techniques of the CPM-procedure (PVMs, class diagrams, and CRC-cards), has central storage of data, supports network distribution of data, supports multiple user access, integrates the modelling techniques, includes version control, and allows integration with PCSs. Furthermore, they describe the development of a prototype, created in Lotus Notes. This prototype has since been further developed and is today applied by the companies GEA Niro A/S and American Power Conversion A/S (APC). Although the use of the Lotus Notes based documentation system has shown that there are significant benefits from applying such a tool for the maintenance of PCSs [2], much is still to be wanted. From a modelling point of view the documentation system fails to offer support for the elaboration of class diagrams and PVMs, but only includes CRC-cards and a hierarchical list of classes that does not show attributes, constraints, and kind-of structure/generalisation. Consequently, both the mentioned companies, who use the Lotus Notes application, apply other software for the creation of PVMs when making big changes or additions to their existing models.

Based on [17] and interviews with four Danish manufacturing companies who apply configuration systems, Hvam et al. [16] propose an extended list of requirements for a documentation system to support the development and maintenance of PCSs. This includes: a coherent product model, version control, access control, change notification, user-friendliness, web-based access, integration to other software systems, possibility of informal rule expressions, hyperlinks to internal and external files, flexibility, configuration system integration, the use of English as language, and an inexpensive solution. Furthermore, a high level description of a possible architecture of such a documentation system is presented in [16].

3.2 Why has the system not been created?

Having established that seemingly there is a need for a documentation system to support the development and maintenance of PCSs, and research that deals with defining such a system has been produced, it could seem strange that a complete documentation system does not exist at present. Based on the experience (of the first author) from studies of several configuration projects, the following possible explanations are offered:

1) While the creation of a documentation system to support the development and maintenance of PCSs seems to require a significant deal of software development, it is presently unclear how many potential customers there will be. It seems that the companies who show the greatest interest are the ones who have a great need for external documentation of the knowledge base of the PCS. These can typically be characterised by having extensive and complex PCS knowledge bases and having to describe product knowledge that is possessed by others than the ones implementing the knowledge in the PCS. However, the number of these kinds of companies in Danish industry is limited. But if a relatively inexpensive documentation system emerged, companies with less need for external documentation of their PCSs may show greater interest.

2) The companies who apply PCSs apply different modelling techniques when documenting their PCSs. Even those who base their development on the CPM-procedure make individual adaptations. Therefore, creating a system that is adequately flexible to support the different kinds of uses could turn out to be very difficult.

3) Although research concerning the creation of a documentation system [16, 17] specifies which modelling techniques should be included, this research does not in a detailed manner deal with topics like: user interface design, detailed definitions of the included modelling techniques, and how to

handle interrelated models. An important part of the basis for developing a documentation system to support the development and maintenance of PCSs has therefore been missing until recently [18].

3.3 Definition of a documentation system

Addressing the need for adequately flexible and software prepared notation formalisms of PVMs, class diagrams, and CRC-cards in order to include these in a documentation system, Haug and Hvam [18] present such a definition. This includes three main views, namely a CRC-card view, a PVM view, and a class diagram view. In the documentation system it should be possible to switch between these views dependant on what level of detail and what kind of information the users are interested in. The three views are shown in figure 3.

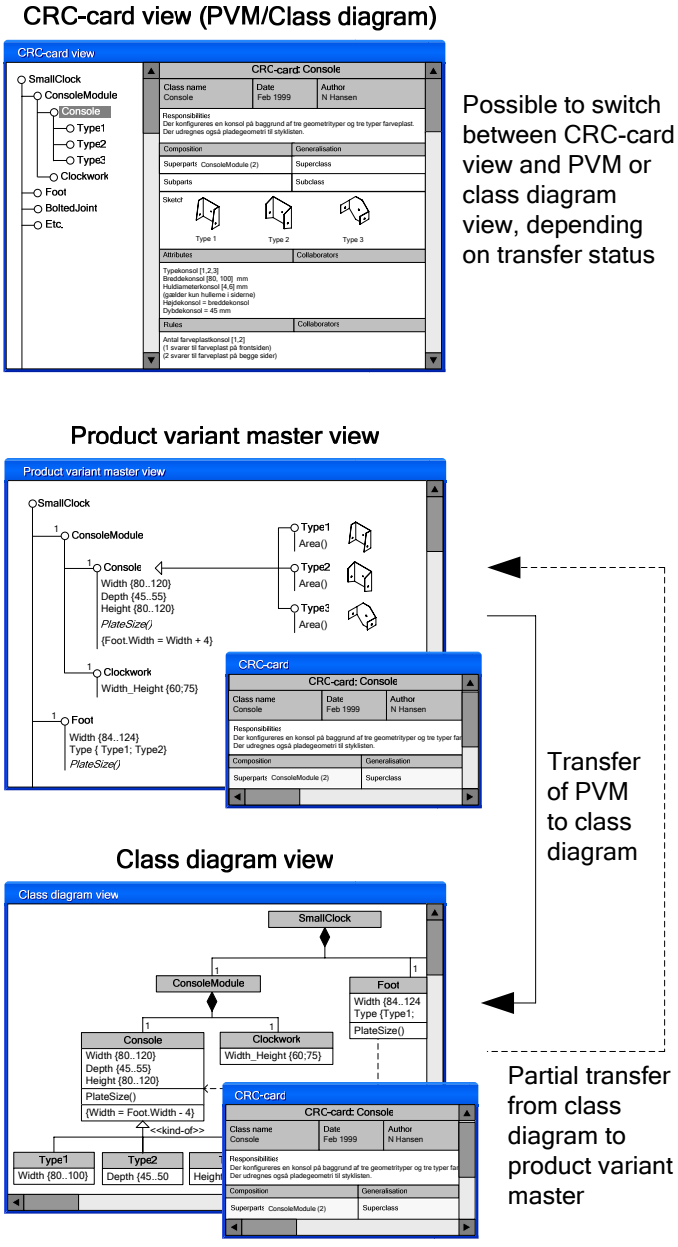


Fig.3. The three main views in the documentation system (adapted from [18])

Yet another step towards the creation of a documentation system that supports the CPM-procedure was taken by Haug and Hvam [19], who, based on the current definitions by CPM [3], present a revised definition of the CRC-card layout, which takes into account future software supported

elaboration of the CRC-cards. The new layout includes several new fields, e.g. for documenting additional relationship types and for organising information about attributes, constraints, and methods. Another extension of the CRC-card layout concerns the inclusion of fields for managing change requests and versions.

4 The creation of a prototype

Besides supporting the elaboration of PVMs, class diagrams and CRC-cards in an integrated fashion, the documentation system should include functionalities similar to the ones of product data management (PDM) systems in order to support the development and maintenance of PCSs in a satisfactory manner. While PDM-related functionalities have been included in numerous software systems, a system which supports and integrates PVMs, class diagrams, and CRC-cards does not exist. Creating a prototype to evaluate this kind of modelling environment was therefore a natural starting point. Based on the definitions in [18, 19], a prototype including the three main views was created. Besides evaluating the defined modelling techniques, another main point was to find a solution principle for handling the model elements which are represented differently or only present in some of the three views.

It was chosen to create the prototype using MS C# .Net, as the Microsoft .Net platform supports Windows software development and because it was considered to ease object-oriented development, offer a development environment that is easy to use, and have good debugging functionality.

As mentioned, an important argument for the creation of a documentation system is to avoid the manual transfers of information between PVMs, class diagrams, and CRC-cards. The three techniques, however, do not only include the same information, but also have individual information, e.g. is the *Sketch/Picture* field of the CRC-card view not included in the class diagram view. One possibility is to create a solution where information is transferred between the three kinds of representations when making changes. This, however, would result in a need for high memory, and redundancy in the stored data. To avoid this, the prototype has only one data model for all three diagrams, which means that the three kinds of representations are different views of the same data. This for instance means that when changing a part of a class diagram model this information would automatically be changed in the corresponding PVM model and CRC-cards if these include this specific type of information. The data model of the prototype is depictured in figure 4, where the attributes and methods are not shown due to lack of space.

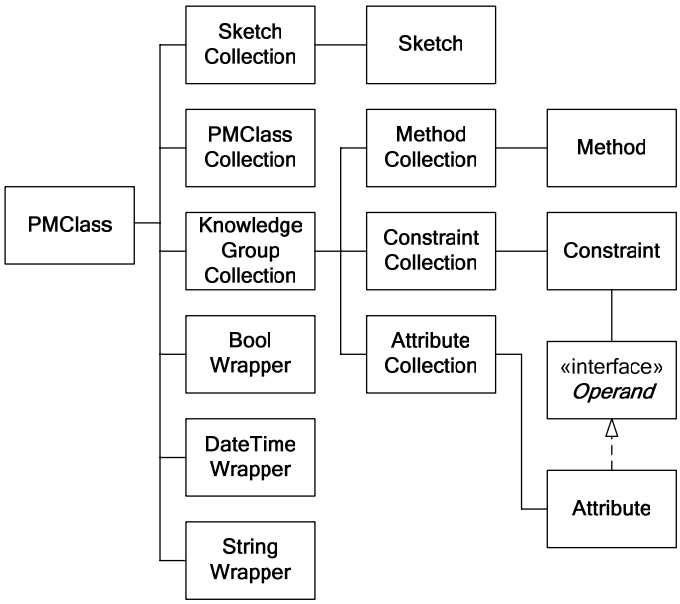


Fig. 4. The common data model

While the three views of the prototype are bound to the data model, the data-model on the other hand,

is not bound to any of the views. It would, therefore, be possible to add, change and remove views of the prototype without adding or deleting classes in the data model.

In its current form the prototype displays the same user-defined classes and relationships in the PVM view and the class diagram view. However, in a further developed version of the documentation tool it should be possible to associate visual content of the data model to particular views. This would among other things be useful in contexts where the PVM view is used for creating an analysis model, and where the class diagram view is used for creating the corresponding design model. For instance, an analysis model could include classes that are not going to be implemented, for which reason these should not be part of the design model. The other way around, the design model could include implementation-oriented classes that are irrelevant in the analysis model, which, therefore, should not be shown in the PVM view.

CRC-cards can be accessed from any of the three views; in the CRC-card view by clicking on a class in the hierarchical list, and in the PVM view and class diagram view by double-clicking on a class. In figure 5 the CRC-card view of the prototype is shown. The CRC-card view consists of a hierarchical list together with a chosen CRC-card. In the hierarchical list, aggregation and inheritance are shown by different symbols, a white node if it is an aggregation class and a black node if it is a specialisation class.

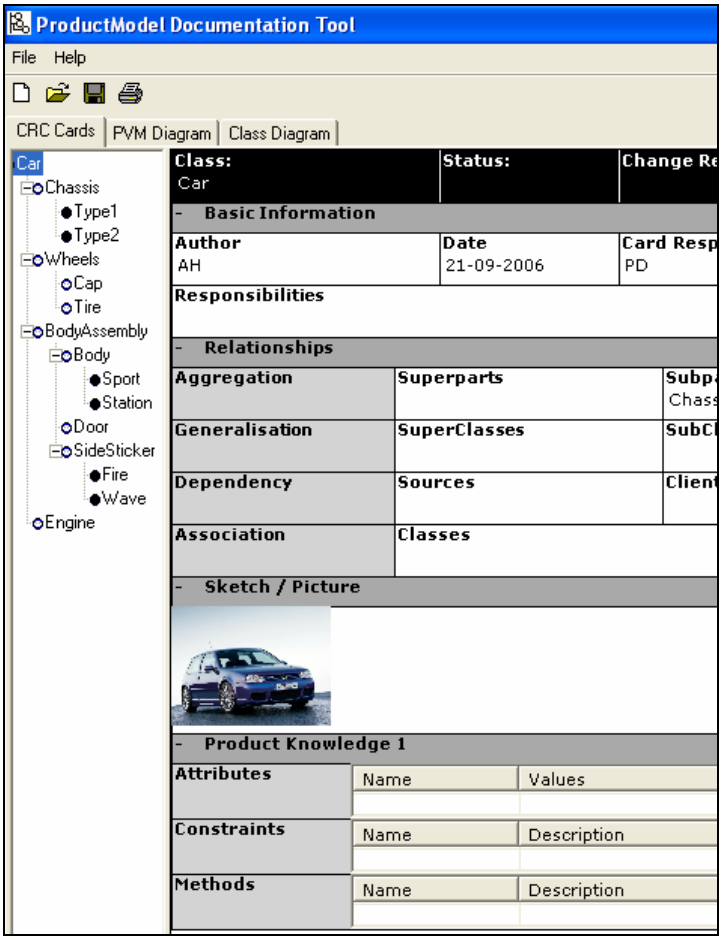


Fig.5. CRC-card view (part of the window)

In the hierarchical list of the CRC-card view classes can be moved, copied or linked by using the standard MS Windows functionalities of drag-and-drop and context menus. Moving a class from one place to another in the hierarchical list means a change in the class' relationships with other classes. This kind of change is automatically updated in the relationship-fields of relevant CRC-cards and on the matching PVM and class diagram. When copying a class, an identical copy is created, which can be pasted anywhere in a given model. A copied class represents a new class in a model, for which

reason it must be given a new and unique name. Changes done to the original class or the copied class does, therefore, not affect the opposite. A class may also be linked to another class. Thereby the same class appears several times in a model. Using linked classes means changes to the class at one place in the product model automatically affect the linked classes. This can be useful when a model includes components that are parts of different assemblies, and where these should hold the same information at every place they appear in the model. For instance, if a class *Screw* appears several times in a model, and new screw-dimensions must be introduced, this only has to be done once. In figure 6 and figure 7 the PVM view and the class diagram view is shown.

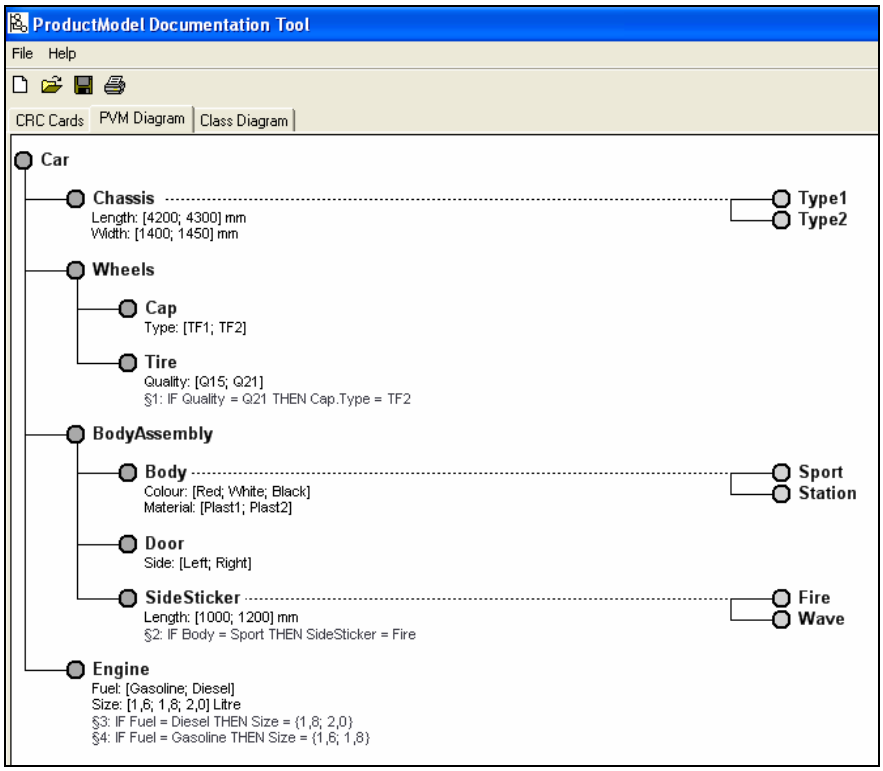


Fig.6. PVM view

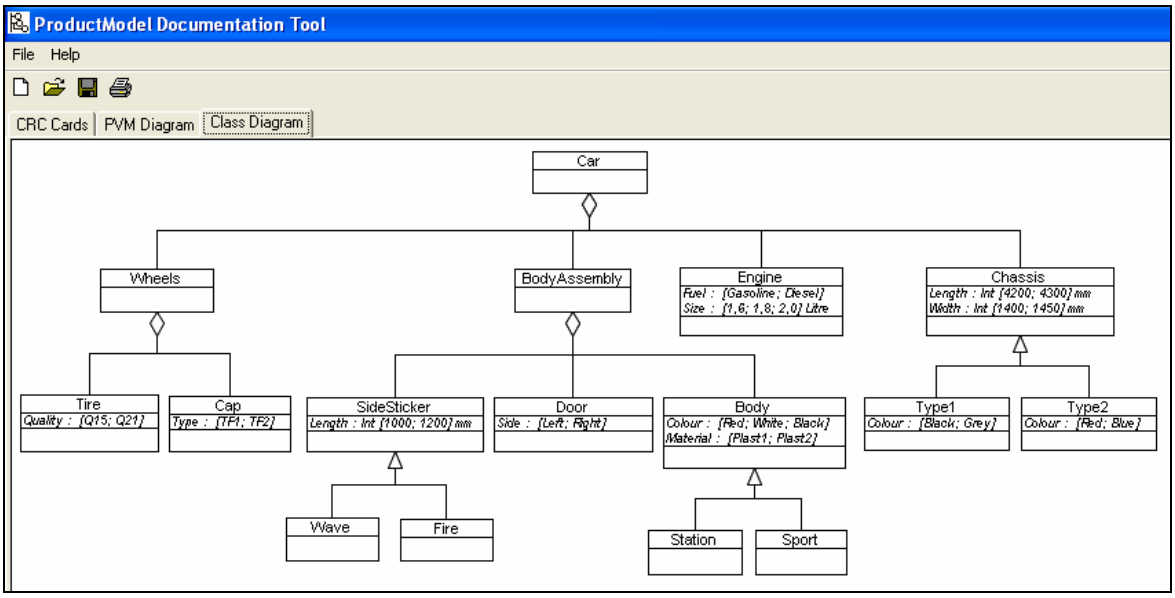


Fig.7. Class diagram view

The elements of the class diagram are created automatically when defining a model in either the CRC-card view or the PVM view. However, placing the classes must be done manually by the user, while the system remembers this the next time the class diagram view is accessed. To increase the user friendliness of the system, a future version should include placement rules for the class diagram view.

In the class diagram view classes can be moved by dragging them to any desired place in the diagram. It is also possible to move several classes at the same time by dragging a box around them. Resizing of classes can be done by dragging the corner points, which are shown when the cursor moves over a class. Like in the PVM view, classes can be created by right-clicking on a class and selecting to insert a new class with a chosen relationship type.

The prototype is capable of saving a created model to a file, as well as printing out the diagrams and CRC-cards of the different views.

The prototype was developed from February to May 2006, and approximately 200 hours were spent designing the software architecture and programming the prototype.

5 Evaluation of the prototype

To evaluate the prototype, two kinds of investigations were carried out. Firstly, data from an ongoing configuration project was put into the prototype to investigate possible limitations of the modelling environment of the prototype. Secondly, the prototype was presented to two companies in order to compare the prototype with their current documentation software and to get other kinds of feedbacks.

5.1 Testing the prototype

The PCS-project, which provided the product model that was put into the prototype, concerns the creation of a PCS to support the creation of tender documents, manufacturing drawings, bill of materials etc. in projects concerning balconies for existing buildings. While the project is ongoing, the company wishes to remain anonymous. The product knowledge in the ongoing project was represented in PVMs that had been created on the basis of the notation shown in figure 1, but with some minor extensions.

As regards the CRC-card view, the experiment showed a need for better possibilities for typing in comments, wherefore more such fields should be included in a final documentation system. In the models from the ongoing project many of the constraints were represented in table form. As the prototype at its current stage only supports the writing of textual expressions, the constraints represented in tables had to be transformed. To avoid this, the final documentation should support the use of tables.

The PVM view allowed putting in the model information from the ongoing PCS-project. However, in the PVM models from the ongoing PCS project, constraints and comments were not placed below classes as defined in the prototype, but were instead placed in boxes near classes. Although the ones, who are going to use the system, properly could be forced to place this kind of information below classes, it seems reasonable to allow the use of boxes in a final documentation system.

The class diagram view was also capable of holding the information from the PVM models from the current case. However, in a final documentation system it should be possible to: turn off some of the contents of a model, insert boxes for comments or constraints, and apply more types of class diagram relationships, as defined in [18].

5.2 Presenting the prototype to users

The prototype was presented to two Danish manufacturing companies, who both have several years of experience with the use of PCSs. The two companies were chosen due to the extensiveness and complexity of their PCSs, as this often implies a need for the creation of external documentation in order to overview the implemented product knowledge.

The first company the prototype was presented to was GEA Niro A/S. Niro is part of the GEA group and is an international engineering company that has a leading market position within the area of design and supply of spray drying plants. Niro primarily use their PCS to support the elaboration of

tenders and, as mentioned, apply a Lotus Notes solution for documenting the knowledge that is implemented in the PCS. Sometimes PVMs, drawn in MS Visio, are used for knowledge acquisition. This use both includes the creation of new PVM models and the recreation of models based on the information in the documentation system, when this needs to be changed. Therefore, both manual transfers of information from PVMs to the documentation system and the other way around occur. Further descriptions of the PCS-project at Niro can be found in [2].

After having seen the prototype, Niro expressed that the use of a further developed version of the prototype most likely could produce several benefits compared to their existing solution. The primary benefit would be the possibility of showing model information in PVMs and class diagrams, which would save Niro time, compared to having to elaborate PVMs manually based on the information in their documentation system. Another benefit would be the possibility of easily creating graphical models, which was considered to be an aspect that could improve the communication with domain experts. On the other hand, Niro requested that the final documentation system would include a range of PDM-related functionalities, such as being able to screen off users with limited modelling prerequisites from some functionalities and information during certain stages of a project. Another important request was that the system should support the use of tables for describing constraints.

The second company to which the prototype was presented was F.L. Smith (FLS). FLS is part of the Danish FLS Industries, and is an engineering and industrial company with an leading market position within the area of development and manufacturing of cement plants. FLS apply a PCS for the creation of budget quotations. FLS document their PCS by using MS Visio, Word, and Access. Compared to Niro, the documentation of the PCS of FLS is only used by the knowledge engineers/system developers. Further descriptions of the PCS-project at FLS can be found in [2, 20].

Having been presented to the prototype, FLS expressed that they might be interested in a similar solution, as this would be likely to enhance the overview of the implemented knowledge and make their PCS easier to further develop compared to the use of their existing methods. Of additional requirements, FLS expressed that they would like to have import/expert functionality so that the documentation system could be interfaced to their PCS, e.g. to allow that class structures and attributes from the documentation system are imported into a PCS, and rules in a PCS are imported into the documentation system.

6 Conclusion

The issue of documentation in PCS-projects is a topic that has been investigated in recent years. This research indicates a need for a documentation system that can provide easily understandable descriptions of what should be or is implemented in a PCS. In this paper the creation and evaluation of a documentation system prototype were described.

A procedure for the development and maintenance of PCSs, which includes three main modelling techniques, has been applied in several cases in Danish industry. However, no software currently exists that supports these techniques in an integrated fashion, which means that manual transfers of information between models are required. Research concerning functional requirements for such a documentation system has been carried out, and recently two papers, which in a detailed manner define the modelling techniques to be included, have been published. Based on these papers, a prototype was created. The aim of the prototype was to evaluate the definitions of the modelling environment, and not PDM-related functionalities. This choice was taken based on the fact that no software includes this kind of modelling environment as opposed to PDM-related functionalities, which are included in numerous software systems. The prototype was created by using C# .Net, and approximately 200 hours were spent on design of software architecture and programming.

The prototype was evaluated by putting a PVM model from an ongoing PCS-project into the prototype and by presenting it to two companies. The experiment of putting a PVM model from an ongoing PCS-project into the prototype showed that the prototype was capable of holding most of the information of this model. However, there were needs for: better possibilities of stating comments in the CRC-cards, support of constraints being formulated in tables, and it being possible to place boxes for comments and constraints next to classes of PVMs and class diagrams. The presentations of the prototype to the two companies indicated that the use of the prototype compared to the use of existing software in many ways could improve the quality of their documentation and save resources in the

creation process. On the other hand, more PDM-related functionalities would be required in order to be a real alternative, just as the possibility of import/export to/from a PCS is an important request.

The development of the prototype indicates that the creation of the modelling environment of a documentation system is a task that can be done within reasonable time limits. The evaluation of the prototype indicates that a less complete system might be adequate, as this could still be a far better solution than existing technologies.

The creation and evaluation of the prototype have provided an important basis for the development of a complete documentation system, since many of the created solutions would be reusable, and because of the feedback the prototype has produced. The prototype developed is therefore, an important step in the direction of creating a final documentation system. If this system meets its expectations, this could have a great impact on the way in which the development and maintenance of PCSs are approached and contribute to higher success rates of such projects.

References:

[1] T. Soininen, J. Tiihonen, T. Männistö, and R. Sulonen, Towards a general ontology of configuration, *AI EDAM-Artificial Intelligence for Engineering Design Analysis and Manufacturing*, Vol.12, No.4, 1998, pp. 357-372.

[2] L. Hvam, A Multi-perspective approach for the design of Product Configuration Systems - An evaluation of industry applications, *Proceedings of PETO Conference*, Lyngby, Denmark, Department of Manufacturing Engineering and Management, Technical University of Denmark, 2004.

[3] L. Hvam, N.H. Mortensen, and J. Riis, *Produktkonfigurering* (Preliminary edition, first edition is to appear in 2006), [Product configuration], Copenhagen, Denmark: Nyt Teknisk Forlag, 2006.

[4] J. Riis, *Fremgangsmåde for opbygning, implementering og vedligeholdelse af produktmodeller - med fokus på konfigureringssystemer*, [Procedure for building, implementing and maintaining product models - with focus on configuration systems], PhD thesis, Department of Manufacturing Engineering and Management, Technical University of Denmark, 2003.

[5] C. Forza & F. Salvador, Managing for variety in the order acquisition and fulfilment process: The contribution of product configuration systems, *International Journal of Production Economics*, Vol.76, No.1, pp. 87-98, 2002.

[6] C. Forza, A. Trentin, and F. Salvador, Product Information Management for Mass Customization: the Case of Kitting, *Proceedings of MCPC2005*, 18-21 Sept. in Hong Kong, 2005.

[7] D. Sabin and R. Weigel, Product Configuration Frameworks - A survey, *IEEE Intelligent Systems & Their Applications*, Vol.13, No.4, pp. 42-49, 1998.

[8] B. Hansen, J. Riis, and L. Hvam, Specification process reengineering: concepts and experiences from Danish industry, *Proceedings of the 10th ISPE international Conference on Concurrent Engineering: Research and Applications*, Madeira, Portugal, July 26-30, 2003.

[9] K. Edwards, and K. Ladeby, Framework for Assessing Configuration Readiness, *Proceedings of the 3rd Interdisciplinary World Congress on Mass Customization and Personalization* (MCPC2005), Hong Kong, Sept. 18-21, 2005.

[10] L. Hvam, *Application of product modelling – seen from a work preparation viewpoint* (Trans.), PhD thesis, Lyngby, Denmark, Department of Industrial Management and Engineering, Technical University of Denmark, 1994.

[11] U. Harlou, *Developing product families based on architectures: Contribution to a theory of product families*, Unpublished dissertation, Lyngby, Denmark, Department of Mechanical Engineering, Technical University of Denmark, 2005.

[12] OMG, *Unified Modeling Language: Superstructure* (Version 2.0: Formal/05-07-04), www.uml.org, 2005.

[13] M. Fowler, *UML Distilled* (3rd edition), Boston, MA, Addison-Wesley, 2005.

[14] K. Beck, and W.A. Cunningham, A laboratory for teaching object-oriented thinking, *SIGPLAN Notices*, Vol.24, No.10, pp. 1-6, 1989.

[15] K. Edwards, L. Hvam, J.L. Pedersen, M. Møldrup, and N. Møller, *Udvikling og implementering af konfigureringssystemer: Økonomi, Teknologi og Organisation*, [Development and implementation of configuration systems: Economy, Technology and Organisation], Final report from research project, Department of Manufacturing Engineering and Management, Technical University of Denmark, 2005.

[16] L. Hvam, S. Pape, K.L. Jensen, K.L., and J. Riis, Development and maintenance of product configuration systems - Requirements for a documentation tool. *International Journal of Industrial Engineering*, Vol.12, No.1, pp. 79-88, 2005.

[17] L. Hvam, and M. Malis, A Knowledge Based Documentation Tool for Configuration Projects, *Proceedings of World Congress on Mass Customization and Personalization*, Hong Kong, Oct. 1-2, 2001.

[18] A. Haug, and L. Hvam, The modelling techniques of a documentation system that supports the development and maintenance of product configuration systems, *Proceedings of IMCM'06*, Hamburg, Germany, June 22-23, 2006.

[19] A. Haug, and L. Hvam, CRC-cards for the development and maintenance of product configuration systems, *Proceedings of IMCM'06*, Hamburg, Germany, June 22-23, 2006.

[20] L. Hvam, Mass Customization for Process Plants, *Proceedings of MCPC2005*, Hong Kong, Sept. 18-21, 2005.